AD-A280 472

1

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DTIC
ELECTE
JUN 22 1994
S G D

SACS: A CACHE SIMULATOR INCORPORATING TIMING ANALYSIS WITH BUFFER AND MEMORY MANAGEMENT

by

William G. Smith

March, 1994

Thesis Advisor: Douglas J. Fouts

Approved for public release; distribution is unlimited.

94-19105

26298

94 6 22 027

Best
Available
Copy

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March, 1994	Master's Thesis

4. TITLE AND SUBTITLE

SACS, A CACHE SIMULATOR INCORPORATING TIMING ANALYSIS WITH BUFFER AND MEMORY MANAGEMENT

5. FUNDING NUMBERS

6. AUTHOR(S)

Smith, William, G.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

SACS is a cache simulator that provides the user with a wide range of timing information, in addition to providing typical information such as hit and miss rates. The SACS model includes read and write buffers, main memory, and cache memory. In addition, SACS supports a number of buffer and data forwarding policies, as well as the traditional block replacement, write, and write miss policies. SACS also includes a self-testing mode which can be used to debug the program after source-code modification.

14. SUBJECT TERMS

SACS, Cache Memory, Cache Memory Simulation, Computer Architecture, Computer Architecture Simulation

15. NUMBER OF PAGES

264

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

SACS: A Cache Simulator Incorporating Timing Analysis
with Buffer and Memory Management

by

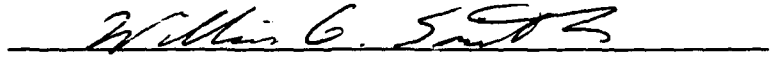
William G. Smith
Lieutenant, United States Naval Reserve
B.S., Saint Bonaventure University, Saint Bonaventure, New York, 1984

Submitted in partial fulfillment
of the requirements for the degree of

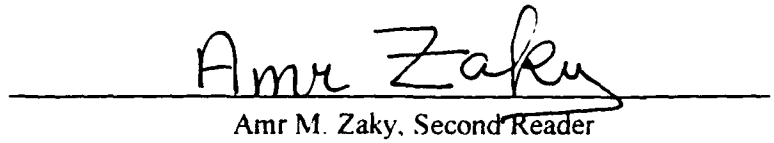
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

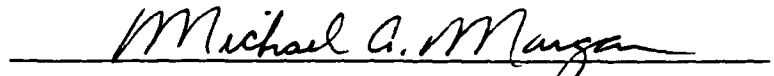
from the

NAVAL POSTGRADUATE SCHOOL
March 1994

Author: 
William G. Smith

Approved by:
Douglas J. Fouts, Thesis Advisor


Amr M. Zaky, Second Reader


Michael A. Morgan, Chairman,
Department of Electrical and Computer Engineering

ABSTRACT

SACS is a cache simulator that provides the user with a wide range of timing information, in addition to providing typical information such as hit and miss rates. The SACS model includes read and write buffers, main memory, and cache memory. In addition, SACS supports a number of buffer and data forwarding policies, as well as the traditional block replacement, write, and write miss policies. SACS also includes a self-testing mode which can be used to debug the program after source-code modification.

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

TABLE OF CONTENTS

I. INTRODUCTION 1
- A. CACHE MEMORIES 1
- B. PROBLEMS OF CACHE MEMORIES 3
- C. EXISTING CACHE SIMULATORS 4
- D. PROBLEMS WITH EXISTING CACHE SIMULATORS 4

II. INTRODUCTION TO SACS 6
- A. THE NEED FOR STILL ANOTHER CACHE SIMULATOR 6
- B. COMPARING SACS TO OTHER CACHE SIMULATORS 6
- C. THE CAPABILITIES OF SACS 7

III. SACS INPUT PARAMETERS 8
- A. INTRODUCTION 8
- B. SIZE ARGUMENTS 9
- C. CACHE ACCESS, HIT, AND MISS ARGUMENTS 9
- D. MEMORY ACCESS AND TRANSFER TIME ARGUMENTS 9
- E. BUFFER ARGUMENTS 9
- F. CACHE POLICY ARGUMENTS 10
- G. SEARCH BUFFERS AND UPDATE BUFFER ARGUMENTS 11
- H. REMOVE READ DUPLICATES AND WRITE DUPLICATES ARGUMENTS 11
- I. PRIORITY ARGUMENTS 11
- J. SACS CONTROL ARGUMENTS 11

IV. SACS DISPLAYS 15
- A. TRACE DISPLAY 15
- B. RESULTS DISPLAY 20
- C. STALL DISPLAY 21
- D. CACHE ARGUMENTS DISPLAY 22
- E. GO TO A SPECIFIC TIME 22
- F. INCREMENT TIME 23
- G. DECREMENT TIME 23
- H. HELP DISPLAY 23

V. SACS DESIGN 25
A. OVERALL STRUCTURE OF SACS 25
B. MAIN EVENT LOOP 25
C. CACHE MODEL 26
D. MEMORY MODEL 34
E. TIME ESTIMATES 38
VI. PROGRAM VALIDATION 40
A. TESTING SACS 40
B. CHECKING COMPLETION TIME 42
C. CHECKING GLOBAL VARIABLES 43
VII. SAMPLE RUNS 44
A. EXAMPLE SACS SIMULATION RUN 44
VIII. CONCLUSION 55
LIST OF REFERENCES 57
APPENDIX (SOURCE CODE FOR SACS) 58
BIBLIOGRAPHY 256
INITIAL DISTRIBUTION LIST 257

I. INTRODUCTION

A. CACHE MEMORIES

Cache memories are usually small memories that contain blocks of data and/or instructions [Ref. 1]. Each block is a copy of consecutive data and/or instruction bytes from main memory. Caches are usually much faster than their main memory counter parts. They are, in effect, short term memory. The cache contains a set of blocks recently accessed by the CPU. The cache can provide the most recently used data and/or instructions to the CPU in less time than it would have taken to get the information from main memory. However, if the cache fails to provide the information, then it fetches the information from main memory and, depending on its design, may choose to enter the information into the cache. The cache also performs memory updates in the form of writes. Memory updates can occur when the CPU makes a write request, or the cache can store new data in the cache blocks and wait until later to write the data to memory [Ref. 2:p. 197]. A dirty block is a cache block which contains data that is more current than memory because of a recent write request.

The concept of the cache storing recently used memory blocks is very simple. In fact, many early cache designs were simple implementations of the concept. However, the task of building an optimized cache is not trivial. The difficulty comes from trying to provide the CPU with the correct data or instruction as soon as it is available. For example,

if a block read is in progress and there is enough data available to satisfy the pending load, the data should be forwarded to the CPU rather than waiting for the block read and cache update. The cache often performs block management between requests. One example of block management is writing dirty data to memory. Another example is reading data that was not part of the CPU request, but located in the same block. This allows the cache to contain all the data that was in the block. An optimized cache performs block management only after completing the CPU request. Complications arise when the CPU makes another request before the block management for the last request is complete. If this happens, the cache has to search for data in the control sections of the cache as well as the cache memory.

More and more cache designs incorporate read and write buffers in their control sections. These buffers allow the cache to perform block management while minimizing the effect on the CPU request [Ref. 3]. For example, if a CPU read request results in a miss and the block victim has a dirty sub block, then writing the dirty sub block to memory may occur after reading the replacement block. This allows the cache to forward the read data to the CPU before writing the old data to memory. Figure 1 provides a simplified data flow diagram that illustrates buffer use in modern cache designs.

Scoreboarding is a term that has been used to describe the process of searching, and choosing the correct value of a register argument. Scoreboarding in a cache represents the act of searching and altering buffers based on new CPU requests. One example of scoreboarding is searching a buffer for CPU requested data. Searching block and write

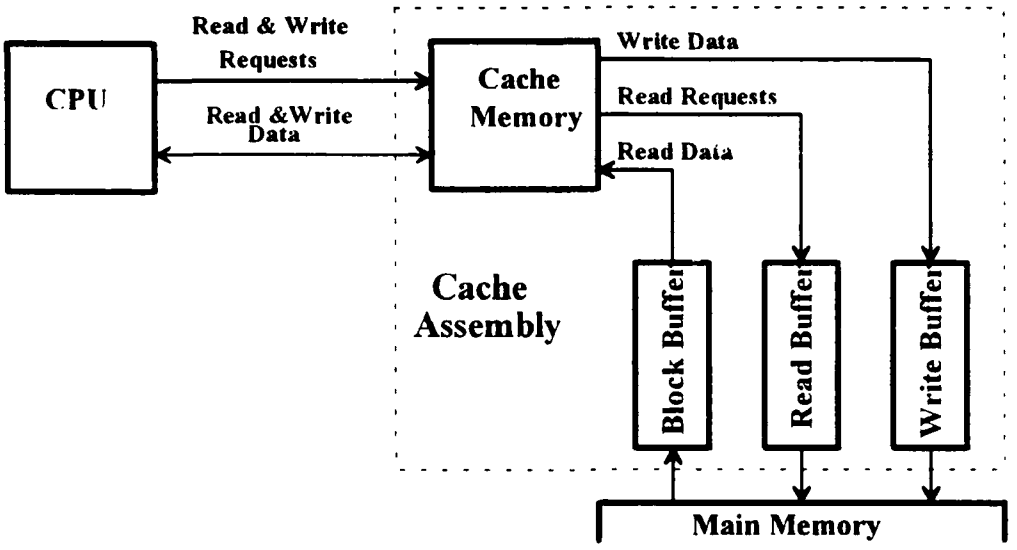


Figure 1 Typical Cache Data Flow Design

buffers may provide data that is not available in the cache memory. Searching the read buffer ensures that no duplicate read requests are placed in the buffer. However, it will not provide any new data. Another example of scoreboarding is updating read requests in the read buffer with data provided by CPU write requests.

B. PROBLEMS OF CACHE MEMORIES

Cache memories can cause more problems than they solve and they can significantly complicate the memory model. The most obvious problem with a cache memory is that it does not always contain the data that the CPU requests. This is defined as a read miss. A write miss is when the CPU makes a write request and the correct block is not in the cache. Conversely, a read hit is when the data is available, and a write hit is when the block is available. There are three types of cache misses [Ref. 4:p. 419]. First,

the *compulsory miss* results when data is accessed for the first time. Second, the *capacity miss*, occurs when the cache is not big enough to carry all the blocks required. The third type, *collision misses*, occurs when the cache requires several main memory blocks that map to the same set of cache blocks. This causes a form of thrashing similar to that seen in virtual memories.

In some numerical calculations, the cache makes the average access time greater than the main memory access and transfer times. This is often due to matrix operations that access elements across rows and force the cache to enter an entire block of data for each element read. Depending on the size of the matrix, it may not be possible to save enough blocks to ensure that the next block accessed was not selected as a victim and replaced. As a result, many architectures support special load and store instructions that bypass the cache.

C. EXISTING CACHE SIMULATORS

There are a number of cache simulators. Two examples include Dinero III, and Tyco [Ref. 5]. Dinero III provides hit and miss data for a wide range of input arguments. Dinero III will also simulate either a unified cache, or separate data and instruction caches. Tyco, on the other hand, simulates several different cache options simultaneously for comparison.

D. PROBLEMS WITH EXISTING CACHE SIMULATORS

Unfortunately, both Dinero III and Tyco limit their simulations to hit-miss calculations. With nothing but hit-miss information, the designer cannot optimize his cache

for the lowest average access time. Most caches are designed to have low average access times rather than high hit rates [Ref. 4:p. 405]. Since Dinero III and Tyco do not perform any timing analysis, they may mislead the designer. Dinero III and Tyco also do not provide any buffer simulations because they assume that the cache has all the time it needs between loads and stores to complete all of its block management. Since buffer management and scoreboarding have such a large effect on the average access time, there is an obvious need for a simulator that can perform accurate timing analysis, buffer management, and scoreboarding.

II. INTRODUCTION TO SACS

A. THE NEED FOR STILL ANOTHER CACHE SIMULATOR

A cache simulator should not only simulate the cache memory, it should simulate main memory and any buffers it uses. As discussed earlier, neither Dinero III nor Tyco provide any means for simulating buffers or memory.

Without timing analysis, the designer is unable to determine the effect of scoreboarding protocols. These protocols, which are usually very difficult to implement, can be avoided by delaying any read requests until all writes are completed and the last read block has been entered into the cache. Timing analysis allows the designer to choose the scoreboarding technique that best suites his or her resources and architectural requirements. A hit-miss cache simulator reduces this process to guess work.

B. COMPARING SACS TO OTHER CACHE SIMULATORS

As previously discussed, other cache simulators provide hit-miss results, while SACS provides the all important average access time. However, in addition to the correct performance measurements, a simulator should illustrate a cache's strengths and weaknesses. It should give the user a clear understanding of how to improve the cache's design. With Dinero III, the user could only guess on how to improve his/her cache design. Tyco attempted to correct this problem by allowing the user to simulate several caches simultaneously. However, given the number of different variables and policy choices,

exhausting all possible combinations is not the best way to design a cache. SACS is unique because it provides the user with a detailed analysis of exactly what the cache was doing during a simulation run. The user can then identify and correct specific problems with a cache design.

C. THE CAPABILITIES OF SACS

SACS allows the designer to experiment with different policies while measuring their affect on the average access time. SACS provides the user with more detailed information because it maintains a log of how every clock cycle is spent. This log is kept in the form of a histogram. It allows the user to see exactly how much time is spent performing read or write requests. It also records how many times a request was completed within a given time period. With these details, the user can easily evaluate the cache's performance. A second histogram is available which details the amount of time spent performing cache accesses, memory accesses, and waiting for full buffers. With this histogram, the designer can target specific weaknesses. It also provides a good comparison of the effect of different scoreboarding policies between runs.

III. SACS INPUT PARAMETERS

A. INTRODUCTION

SACS provides a wide range of input parameters to model various different functionally diverse caches. While it is impossible to imagine what kinds of caches designers might build in the future, every effort was made to allow the designer to simulate, or most nearly approximate, his or her design. Table 1 lists arguments that SACS supports.

TABLE 1.
SACS INPUT ARGUMENTS

Cache Size (-cs n)	*Read Forward (-rf -drf)*
Blocks Size (-bs n)	*CPU Waits For Cache Writes (-cwfcw -dcwfcw)*
Sub Block Size (-sbs n)	*Search Block Buffer (-sbb, -dsbb)*
Associativity (-a n)	*Update Read Buffer (-urb, -durb)*
Word Size (-ws n)	*Remove Read Duplicates (-rrd, -drrd)*
Read Cache Access Time (-rcat n)	*Remove Write Duplicates (-rwd, -drwd)*
Read Cache Hit Time (-rcht n)	*Read Priority (-rpr n)*
Read Cache Miss Time (-rcmt n)	*Write Priority (-wpr n)*
Write Cache Access Time (-wcat n)	*Read For Write Allocate Priority (-rfwapr n)*
Write Cache Hit Time (-wcht n)	*Write Dirty Block Priority (-wdbpr n)*
Write Cache Miss Time (-wcmt n)	*No Priority (-npr n)*
Memory Access Time (-mat n)	*Trace (-t, -dt)*
Memory Transfer Time (-mtt n)	*Check (-c, -dc)*
Buffer Cache Access Time (-bcat n)	*Test (-test)*
Read Buffer Size (-rbs n)	*Key Board IO (-kbio, -fio)*
Write Buffer Size (-wbs n)	*Data File Name (-f "File Name")*
Block Replacement Policy (-brp e1)	*Screen Histogram Max Index (-shmi n)*
Write Policy (-wp e2)	*File Histogram Max Index (-fhmi n)*
Write Miss Policy (-wmp e3)	

n : Unsigned Integer
e1 : enumeration type (LRU, FIFO, RAND)
e2 : enumeration type (Write Through, Write Back)
e3 : enumeration type (Write Around, Write Allocate)

B. SIZE ARGUMENTS

All sizes are entered in bytes. *Cache Size* must be an even multiple of both *Block Size* and *Associativity*. *Block size* must be an even multiple of *Sub Block Size*, and *Sub Block Size* must be an even multiple of *Word Size*. *Word Size* may be any positive integer that does not force an integer overflow in *Cache Size* or *Block Size*.

C. CACHE ACCESS, HIT, AND MISS ARGUMENTS

Cache Access Times represent the time required for a request to access the cache, providing the cache is not busy. At the end of this time it is determined whether the request is a hit or a miss. The *Cache Hit Time* represents the time required to complete a request once the cache access time has expired and the request has been identified as a hit. If the request was a miss, then the *Cache Miss Time* is used instead.

D. MEMORY ACCESS AND TRANSFER TIME ARGUMENTS

The *Memory Access Time* is the time required for a buffer to access, and then transfer, the first word of a request from memory. *Memory Transfer Time* is the time required for a buffer to transfer each consecutive word following the first access.

E. BUFFER ARGUMENTS

After a memory read, the *Block Buffer* contains the new data that must be entered into the cache. The time that the *Block Buffer* takes to access the cache is called the *Buffer Cache Access Time*. The *Block Buffer* may have to wait longer because the cache is busy. The buffer cache access will not occur if a read or write cache access is required during the same clock cycle. However, once the access begins, the read and write cache accesses are

locked out until the cache is updated. The *Read* and *Write Buffer* sizes can be any positive non zero integer (<100).

F. CACHE POLICY ARGUMENTS

Block replacement policy determines the method used to choose the location of a new block in the cache. SACS supports three block replacement policies: Least Recently Used (*LRU*), First In First Out (*FIFO*), and Random (*RAND*). There are two write policies: *Write Through*, and *Write Back*. The *Write Through* policy forwards the data to memory immediately after a write request. However, in the *Write Back* policy, the data is saved in the cache until the block that contained the data is selected as a victim. Dirty bits indicate which sub blocks have new data that must be written to main memory.

SACS can easily be modified to support new write, or write miss policies by adding the new policy name to *Write Policy Types*, or the *Write Miss Policy Types* in *SACS.h*. The code to simulate these new policies must be placed in the procedures *Write Hit*, and/or *Write Miss* which are both located in *Cache.c*. New block replacement policies may also be added by modifying *Replacement Policy Types* in *SACS.h*, and *Select Block Victim* in *Cache.c*.

If CPU Waits For Cache Writes is asserted the CPU will wait after a write request until the cache is complete with the write. Otherwise, it is assumed that the cache can carry out the write while the CPU continues with other instructions.

If *Read Forward* is asserted, then a read miss is complete once the data required arrives in the block buffer. Otherwise, the read must wait until the block updates the cache.

G. SEARCH BUFFERS AND UPDATE BUFFER ARGUMENTS

Search Block Buffer allows the cache to search the block buffer in case the read data was already received from memory. If any part of the data is found in the buffer, then the size of the request will be appropriately reduced. *Update Read Buffer* allows a write memory request to provide data required by a read request. If any read request needs the data provided by the write memory request, then the size of the read request will be reduced appropriately, and the data is not read from memory.

H. REMOVE READ DUPLICATES AND WRITE DUPLICATES ARGUMENTS

Removing read and write duplicates means that requests that have intersecting or concurrent data will get spliced together into one request. Otherwise, a buffer may contain multiple requests to the same memory location.

I. PRIORITY ARGUMENTS

In SACS, the user specifies the priority of requests. The lowest numbers represent the highest priority. This allows the designer to simulate a cache that must finish all writes before starting a read. It also allows the designer to delay reads for write allocated blocks, and the writing of dirty blocks.

J. SACS CONTROL ARGUMENTS

1. Introduction

SACS has control arguments that allow the user to select one of several modes of operation.

2. Trace Argument

The trace mode permits the user to step through a trace, one clock cycle at a time, viewing the contents of the cache and buffers, and obtain statistical results.

3. Check Argument

When SACS is in the check mode, it performs a self check of all of its global variables. These checks include checking to see that all the global variables that should remain constant, do in fact remain constant. Global variables that are not constant are checked to see that they are within prescribed bounds. This form of checking can occur while the program is in any other mode (i.e., Trace, Test, or Key Board I/O). This check can be performed during normal data runs. This kind of checking helps to identify errors that might have gone unnoticed. It also assures the user that the program did not catastrophically fail during a run.

4. Test Argument

When SACS is in its test mode it can do nothing else. It will automatically generate its own input data. The data is generated by randomly selecting a finite number of test cases to use. Each test case has a combination of seven load/store instructions. The expected number of read and write hits for each test case is known. Therefore, the total number of read and write hits for the trace can be determined. The actual addresses used for each test case are chosen randomly. However, all the loads for a particular test case will map to the same block. Similarly, all stores will also map to the same block. SACS will randomly select its own set of input parameters, ignoring any other arguments entered.

It will then run the main routine on the test trace as if it was a user defined file. Once the trace is complete, SACS will compare the results with what it expected from the random trace. If no errors have been detected, then SACS will repeat the process of randomly generating its input file and input parameters. The test mode places SACS into an infinite testing loop. The only way to terminate the process is to kill the process. The decision not to give the user a more graceful way out of the test mode was made because C does not provide a way to trap IO. While there are operating system methods of trapping IO, they would have made the program system dependent and, therefore, non-portable. The current version of SACS has been compiled and run on both a PC running DOS and a SPARC station running Sun -OS without any changes to the source code.

5. Key Board IO Argument

SACS normally accepts its inputs from a data file. However, it can accept inputs directly from the user. This input mode can be used with the trace and checking modes if desired. SACS will ask for each request from the terminal as required. The trace display will appear each time a new request is made. However, it will not stop every clock cycle unless the user selects the trace mode.

6. Data File Name Argument

SACS normally assumes that the data file is named "SACS.Dat", however the user can specify the name of the file he or she wishes to trace using the data file name argument.

7. Histogram Max. Index Arguments

SACS provides the user with two timing histograms. One provides timing analysis for read and write requests. The first histogram illustrates how many requests were completed during the designated time. The second histogram provides timing analysis for the number of times the cache waited to complete a particular task in the designated time. The maximum index for these histograms is the maximum number of time bins. Since there may be data out of the maximum index range, the last bin is used to total all events with times greater than or equal to the maximum index. The screen histogram maximum index is the number used for screen displays. The screen maximum histogram index has a default value of 4, which allows all displays to fit on a standard 80 column screen. However, if a Unix window is available, the designer may want to raise this number to get more detailed displays. The file histogram maximum index is used for the output file generated by SACS. This file may have a much larger range because it does not have to be printed on a screen. The output file is compatible with MATLAB script files. This allows results to be read and processed by MATLAB for statistical analysis and plotting purposes.

IV. SACS DISPLAYS

A. TRACE DISPLAY

1. Introduction

SACS includes a trace mode that allows users to monitor the behavior of the simulated cache and the implemented policies and scoreboarding techniques. The user may also need to debug any modifications made to SACS. The trace mode is also very useful in identifying any programming errors. The trace mode allows the designer to review the status of the cache at the end of each clock cycle. A trace display is shown in Figure 2.

```
Current Request:     Read                          Time:                46
Address:             2AF769F9                      Next Request Time:   49
Size:                07                            TOA:                 49
Cache Waiting For:   Read Memory Request
Memory Waiting For:  Memory Read Access            Cache Hit:           No
Block Waiting For:   Memory Block Transfer         Buffer Hit:          No

  Set Block  Address  V/D  V/D  V/D  V/D
00031 00124  25E379F0 0 0  0 0  0 0  1 0
      00125  2AF769F0 0 0  0 0  0 0  0 0
      00126  00000000 0 0  0 0  0 0  0 0
      00127  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             2AF769F9     16     07   00125      00           7          53

Write Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             25E37A10     10     00   00000      11           5          58

Block Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             2AF769F8     00     00   00125      00           7          53

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 2 Trace Display

2. Current Request Fields

The trace display contains the last CPU request, as well as the request address and size. SACS can only process requests that are contained within the same block. If a request spans across two or more blocks, then the request is split up into block size requests and processed separately.

3. Cache Waiting For Field

Whatever is holding up the last CPU request is shown in the *Cache Waiting For* field. In Table 2, we can see all the things that the cache might have to wait for. If the cache is waiting for nothing, then the request has been satisfied. If the cache is waiting for a read or write cache request, then the cache is being accessed. If the cache is waiting for a memory request, then some part of the request must be retrieved from memory and the data is not available. If the request can not proceed because one of the buffers is full, then *Cache Waiting For* will indicate whether the request is waiting for the full read buffer or the full write buffer. Note that a read miss request may have to wait for dirty blocks to be written to memory, making it possible for a read request to wait on a full write buffer.

4. Memory Waiting For Field

Memory Waiting For identifies which memory function is in progress. When a new memory access begins, *Memory Waiting For* indicates whether it is a memory read request or a memory write request. Once the access of the first word is in progress, then *Memory Waiting For* will indicate either read access or write access. However, once the first word has been received, then *Memory Waiting For* will switch to indicate either a

read transfer or a write transfer. If a memory read request needs to begin and the block buffer is busy because it has not updated the cache from the last request, then *Memory Waiting For* will indicate cache update.

5. Block Waiting For Field

Block Waiting For describes what the block buffer is doing. It will indicate memory block transfer when a memory read access or transfer is in progress. However, once the transfer is complete, the block buffer must update the cache. If the cache is busy, then *Block Waiting For* will indicate that it's waiting for cache access. When the cache is not busy and the cache update begins, then *Block Waiting For* will switch to indicate block cache transfer.

TABLE 2.
WAIT FOR CONDITIONS

Cache Wait For Conditions	Memory Wait For Conditions	Block Buffer Wait For Conditions
Nothing *Read Cache Request* *Read Memory Request* *Write Cache Request* *Write Memory Request* *Full Read Buffer* *Full Write Buffer* *CPU Cache Access*	*Nothing* *Memory Read Request* *Memory Read Transfer* *Memory Write Request* *Memory Write Access* *Memory Write Transfer* *Cache Update*	*Nothing* *Memory Block Transfer* *Block Cache Access* *Block Cache Transfer*

6. Timing Data Fields

The trace display shown in Figure 2 includes a time field that indicates how many clock cycles have passed since the start of the run. The *Next Request Time* is the

time at which the CPU either has, or will make, another request. The *TOA* field indicates when the next memory word will arrive into the *Block Buffer*. However, if a write was in progress, this field would be a *TOD* field, and would indicate the time that the next word will leave the *Write Buffer*.

7. Cache Hit and Buffer Hit Fields

The cache hit field indicates whether a request is a hit or not. The buffer hit for a read request indicates whether the data requested is in the *Block Buffer*. A buffer hit for a write request indicates that the *Write Buffer* needed to write the data to memory anyway. A buffer hit will only occur during a cache miss. Buffer hits allow the designer to determine how many times the scoreboarding was used during a run to avoid a memory access. However, the true measurement of a cache's performance is its average access time.

8. Address Block Selection Display

During a trace run it is helpful to see the cache set that the request address got mapped to. This includes all the blocks that the cache had to work with to satisfy the request. If the request is a hit, the block that it hit on has to be in this set. If a block victim is chosen, it has to be chosen from this set.

9. Buffer Displays

The contents of all the buffers are displayed so that the designer may see what the memory is working on. The *Request Size* is the number of bytes that have to be read in order for the current request to have all the data originally asked for. This number

might be reduced from the original request size if part of the data is in the cache or if *Block Buffer* is allowed to contribute data to the request. During the trace, the address, size, and request size will constantly change as CPU requests, and memory accesses and transfers, are made. The block field indicates which cache block the read data will be placed in. The priority field indicates the priority of the memory request. A zero priority indicates that the request is in progress. The next lowest number will be serviced next, unless a new memory request is made. If more than one request has the same priority number, then the read buffer will take priority. If the read buffer has more than one request with the same priority, then the priority will switch to FIFO.

10. Time Required and Completion Time Fields

Every memory request has a *Time Required* to complete and a *Completion Time*. The *Time Required* to complete is the time that memory will have to service the request. The *Completion Time* is the time that the request is expected to be removed from the buffer.

11. Next Command Please

The next command line includes a prompt for the user of all the available commands. Shown in Table 3 is a list of all the commands. In the following paragraphs, these commands are discussed in detail.

TABLE 3. TRACE COMMANDS
Trace Display (T) *Results Display (R)* *Stall Timing Display (S)* *Cache Arguments Display (C)* *Go To A Specific Time (G #)* *Increment Time (#)* *Decrement Time (-#)* *Help (H)*

B. RESULTS DISPLAY

The *Results Display*, shown in Figure 2, provides the user with the number of requests, cache hits, and buffer hits. The hit rates are a combination of both the cache hits and the buffer hits. The *Request Time Histogram* gives the number of requests that were completed in the prescribed time. The total amount of time spent on each request, and the average access times, are also displayed. As discussed previously, the average access time is the ultimate measure of cache performance.

Requests Break Down

Request Types	Number of Requests	Number of Cache Hits	Number of Buffer Hits	Hit Rates	Miss Rates
Read	4	1	1	50.00%	75.00%
Write	7	4	0	57.14%	42.86%
Total	11	5	1	54.55%	45.45%

Request Time Histogram.

	Time=00	Time=01	Time=02	Time=03	Time>=04	Total	Ave Access Time
Read	0	2	0	0	2	11	2.750000
Write	3	4	0	0	0	4	0.571429
Idle	0	21	0	0	0	21	

Next Command Please [T, R, S, C, G #, #, -#, Help] >>>□

Figure 3 Results Display

C. STALL DISPLAY

The stall display provides the designer with an exact account of where the cache spent all its time. As shown in Figure 4, the stall time histogram lists all the events that the cache has ever waited for, and the number of times that the cache waited for an event within a designated period.

Stall Time Histogram

	Time=00	Time=01	Time=02	Time>=03	Total
Nothing	4	0	1	3	21
Read Cache Request	0	4	0	0	4
Write Cache Request	0	4	0	0	4
Read Memory Request	0	0	0	2	7
Write Memory Request	0	0	0	0	0
Full Read Burffer	0	0	0	0	0
Full Write Buffer	0	0	0	0	0
CPU Cache Access	0	0	0	0	0

Next Command Please [T, R, S, C, G#, #, -#, Help] >>>□

Figure 4 Stall Timing Display

D. CACHE ARGUMENTS DISPLAY

The cache arguments display allows the user to review the arguments used by the simulator. Figure 5 shows an example of the cache arguments display.

```
                              Cache Arguments List

Cache Size:                 00008192      Read Forward:                  Yes
Block Size:                 00016         CPU Waits For Cache Writes:    No
Sub Block Size:             00004         Search Block Buffer:           Yes
Associativity:              04            Update Read Buffer:            Yes
Word Size:                  00004         Remove Read Duplicates:        No
Read Cache Acccess Time:    0             Remove Write Duplicates:       No
Read Cache Hit Time:        0             Read Priority:                 10
Read Cache Miss Time:       0             Write Priority:                11
Write Cache Access Time:    1             Read For Write Allocate:       12
Write Cache Hit Time:       1             Write Dirty Block Priority:    13
Write Cache Miss Time:      0             No Priority:                   100
Memory Access Time:         3             Trace:                         Yes
Memory Transfer Time:       1             Check:                         Yes
Buffer Cache Access Time:   1             Test:                          No
Read Buffer Size:           04            Key Board IO:                  No
Write Buffer Size:          04            Data File Name:                SACS.Dat
Block Replacement Policy:   LRU           Screen History Max Index:      0004
Write Policy:               Write Through File History Max Index:        0010
Write Miss Policy:          Write Allocate

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>□
```

Figure 5 Cache Arguments Display

E. GO TO A SPECIFIC TIME

Because data traces are usually very long, SACS was given the ability to run to a specific time. This allows the user to begin a long trace and inspect the results after a reasonable amount of time. If the user is debugging SACS because of a modification or, God forbid, there is an original error in SACS, this command will allow the user to advance to the last time the error occurred. If the time specified is earlier than the current time, then

SACS will temporally turn off the trace, restart the run from the beginning, and stop at the desired time. Once SACS is at the desired time, it will turn the trace mode back on. To the user, it will appear that SACS went backwards. However, in fact, SACS can not run its simulations in reverse. Obviously, if the *Desired Time* is vary large, then this process could take a great deal of time.

F. INCREMENT TIME

Increment time allows the user to adjust the time using a relative step size instead of an absolute desired time.

G. DECREMENT TIME

Decrement time allows the user to adjust the time using a relative step size instead of an absolute desired time. Again, SACS must restart the run from the beginning to stop at the desired time.

H. HELP DISPLAY

The help menu, shown in Figure 5, gives the user simple descriptions of what all the trace commands can do.

```
                        Help Menu

[T] Trace Display:
    Displays current request, status of memory, and contents
    of buffers.

[R] Results Display
    Displays a break down of read and write cache hits, and
    buffer hits, including a timing analysis.

[S] Stall Timing Display:
    Displays  histogram of the time spent on each stall.
    Stalls represent time delays in completing a request.

[C] Cache Arguments Display:
   Display input arguments to SACS.

[G]   Go:         Go to end of run.
[G #] Go To:      Go to Time #.
[#]   Step:       Increment Time By #.
[-#]  Back Step:  Decrement Time By #.
[H]   Help:       Displays this help menu.

Next Command Please [T, R, S, C, G #, #, -#, Help] >>>
```

Figure 5 Help Display

V. SACS DESIGN

A. OVERALL STRUCTURE OF SACS

SACS simulates all events one clock cycle at a time using a global variable named *Time*. Normally, it is preferable to perform timing simulations using event queues so that time can advance to the next event. However, in most cache simulations, so many events happen in one clock cycle that an event queue would probably not improve the performance of the simulator.

B. MAIN EVENT LOOP

In the main event loop of SACS, *Time* is incremented one clock cycle at a time. *Time* is never changed by any other procedure. The requests are entered into the simulation from *Get Next Request*. Simulation of all events is performed by the *Main Event Loop* calling *Cache Model*, *Memory Model*, and *Update Cache*.

SACS insures that all events that can be started during a particular clock cycle are started, and that all events that can complete during a particular clock cycle do. CPU accesses to the cache are given priority over the block buffer cache updates.

SACS's main loop includes the source code to control testing, checking, and tracing. The *Desired Time* variable is controlled entirely by the *Main Event Loop*. *Desired Time* represents a user request to advance the simulation to a particular time with the trace off. SACS can not run *Time* backwards. However, if the *Desired Time* is less than *Time*,

then *Time* is reset back to zero and the run is repeated up to the *Desired Time*. The user can make time requests using arguments "G #", "#", or "-#".

Throughout *Main Event Loop*, *Cache Waiting For* is checked to see if it's equal to *Nothing*. This indicates that the last request has been serviced and that the cache is ready for the next request. The procedures that model specific events as *Read Hit*, *Read Miss*, and *Access Cache* are called repeatedly during their simulations. They utilize *Cache Waiting For* and *Time* to determine what to do next. If any of these procedures need to wait for a period, either to simulate an access or because a resource is not available, then they will set *Cache Waiting For* to the appropriate value. The modeling procedures in *Memory Model* work the same way using *Memory Waiting For*.

Whenever SACS finds an error or a discrepancy then the boolean variable *Discrepancy Found* is set to *Yes*. This forces SACS into a trace mode so that the user may try to identify the cause of the error. In test mode, a discrepancy forces SACS out of test mode so that the trace file that caused the error is not erased by a new file.

C. CACHE MODEL

Cache Model makes all the necessary calls to simulate cache memory. *Cache Model* decides which calls to make based on the values of *Cache Hit* and *Request*. This function is called every time *Time* is incremented. If there are no read or write requests waiting to be completed, the function does nothing. The value of *Cache Hit* will remain *Unknown* until the appropriate cache access time has expired. Then, *Cache Model* will call *Is Request A Hit* to determine if the request is a hit or a miss.

1. Is Request A Hit

Is Request A Hit determines if the request is a hit or a miss, and sets *Cache Hit* to the appropriate value. *Is Request A Hit* will find the *Set Number* that the data is supposed to be in. Then, all *Cache Block Addresses* in that set will be checked to see if they equal the *Block Address* for that request. If the correct block is found, then all sub blocks that are required to satisfy the request will be inspected for validity. If they are valid then *Cache Hit* will equal *Yes*.

2. Read Hit

Read Hit is called to simulate a cache hit during a read request. *Read Hit* simply finishes simulating the cache access for the hit. *Read Cache Hit Time* is the time required to send the data from the cache to the CPU. Note that the time to locate the block in the cache is simulated in *Cache Model*. *Read Hit* is called repeatedly while *Time* is incremented until *Access Cache* returns with *Cache Waiting For* equal to *Nothing*. *Access Cache* will return *Cache Waiting For* equal to *Read Cache Request* until the *Read Cache Hit Time* has expired.

3. Read Miss

Read Miss is called to simulate a cache miss during a read request. *Read Miss* first simulates the time it would take to perform all the block management for a read miss. This time is called *Read Cache Miss Time*. Once that time has expired, *Read Miss* will call *Select Block Victim* to pick a block in the set. When *Select Block Victim* returns

with *Cache Waiting For* equal to Nothing the *Request Block Number* will contain the new block number where the data will be placed.

Once the new block has been chosen, *Read Miss* will call *Add To Read Buffer*. If *Read Forward* is selected, then *Required Size* for the memory request will be equal to the *Request Size*. However, if it is not, then *Required Size* for the memory request will equal *Block Size*. The *Required Size* in read memory request tells the *Memory Model* how much of the requested data must be read into the *Block Buffer* before resetting *Cache Waiting For* back to *Nothing*. By setting *Required Size* equal to *Block Size*, *Read Miss* is forcing *Memory Model* to read in the entire block before setting *Cache Waiting For* back to *Nothing*. Once the *Memory Model* has received the data, it is assumed to be available to the CPU during that clock cycle.

4. Write Hit

Write Hit is called to simulate a cache hit during a write request. *Write Hit* will first simulate the time to write the data to the *Request Block Number* in the cache. The time to locate the block was simulated by *Cache Model*. Once *Write Cache Hit Time* has expired then *Write Hit* will perform the block management for the request. The block management is dictated by the *Write Policy*. For a *Write Back* policy, the sub blocks written to must have their dirty bits set. This is done by *Set Dirty Bit*. For a *Write Through* policy, the memory request must be entered into the write buffer. This is done by *Add To Write Buffer*.

5. Write Miss

Write Miss is called to simulate a cache miss during a write request. *Write Miss* will first simulate the time needed to make all memory requests. This time is called *Write Cache Miss Time*. This is only the time required to make the requests, not the time required to complete them. The time to determine that the correct block was not in the cache memory was simulated by *Cache Model*. The memory requests are entered into the buffers after the *Write Cache Miss Time* expires. The memory requests are dictated by the *Write Miss Policy*. The simplest policy is *Write Around*. For a *Write Around* policy, the write data is placed in the *Write Buffer* by *Add To Write Buffer*. *Write Allocate*, however, is the toughest simulation in SACS. *Write Miss* must first choose a block to put the new data in. This is done by *Select Block Victim*. Block data not provided by the write request has to be read in. This read request is made by *Add To Read Buffer*. The read address is calculated by adding the request size to the address. Because new address may be in the next block, the *Block Size* may have to be subtracted to make the addition modulo. Sub blocks that were written to in there entirety will have there valid bits set to reflect the presence of the data provided by the CPU. However, if only part of a sub block was written to, then the *Cache Valid Bit* will not be set.

Write Miss uses the *Write Policy* to dictate how the write data is to update the memory. For a *Write Back* policy, dirty bits are set by Set Dirty Bits. For a Write Through, the data is added to the *Write Buffer* by *Add To Write Buffer*.

6. Access Cache

Access Cache is called to simulate the CPU accessing the cache. *Access Cache* first waits for the cache not to be busy. The only reason it could be busy is if the *Block Buffer* is in the process of updating the cache. During this time, *Access Cache* will return *Cache Waiting For* equal to *CPU Cache Access*. Once the cache is not busy then *Cache Busy* is set to *Yes*, locking out the *Block Buffer* from accessing the cache. Then, *Cache Waiting For* will be set equal to *Waiting For Request*. This is a local variable passed by the caller. It will either be equal to *Read Cache Access* or *Write Cache Access*. Then, *Cache Busy* is set for the time specified by Request Time. *Request Time* is a local variable. It could equal any of the hit, miss, or access times. Once *Request Time* has expired then *Access Cache* will set *Cache Busy* equal to *No*, and *Cache Waiting For* equal to *Nothing*.

7. Select Block Victim

Select Block Victim chooses the next block to be used and writes the dirty sub blocks out to the *Write Buffer*. *Select Block Victim* first surveys the cache set that the *Request Address* maps to. The survey includes finding the block that was least recently accessed. This Block Number is stored in *LRUBlock*. Once the set has been surveyed then the *Replacement Policy* dictates how the block is chosen. For the *LRU* policy, *Request Block Number* is set equal to *LRUBlock*. For the *FIFO* policy, *Cache Next Block* keeps track of the next victim block for each set. *Cache Next Block* is initialized to all zeros during the beginning of a run. Therefore, it must be checked to see if it is between the first

and last blocks for the set. If it is not, then *Cache Next Block* for *Set Number* is reset to *First Block*. Once *Select Block Victim* knows it has a valid *Cache Next Block* then *Request Block* is set equal to it. Then, *Cache Next Block* for the *Set Number* is incremented. For *RAND* policy, the block number is chosen randomly from all the blocks in the set.

Select Block Victim writes all dirty sub blocks to the write buffer using *Write Dirty Sub Blocks*. *Write Dirty Sub Blocks* takes care of clearing the dirty and valid bits in the block. *Once Select Block Victim* is called and it gets to the bottom of the function with *Cache Waiting For* equal to *Nothing*, then the *Cache Block Address* for the *Request Block Number* is set equal to the block address of *Request Address*.

8. Set Dirty Bits

Set Dirty Bits sets the dirty bits for all sub blocks that contain data that was modified by a write request.

9. Write Dirty Sub Blocks

Write Dirty Sub Blocks is called to simulate writing all the dirty sub blocks in the *Request Block*. *Write Dirty Sub Blocks* not only clears all the dirty bits, it also clears all the valid bits. *Write Dirty Sub Blocks* prepares a block to receive new data, and is called after a block has been selected as a victim. *Write Dirty Sub Blocks* will search the block for consecutive dirty blocks and splice them together into one write request. The write request is then added to the *Write Buffer*. All of the sub blocks that make up the request will have their dirty and valid bits cleared. This process of searching and writing is repeated until all the bits are not dirty. Then, all the valid bits are cleared.

10. Add To Read Buffer

Add To Read Buffer takes the elements of a request and adds the request to the *Read Buffer*. It will perform all of the searches and updates necessary to support the appropriate scoreboarding protocols.

Add To Read Buffer will begin by searching the cache and *Block Buffer* for each byte in the request, starting at the beginning of the request. Every time a byte is found in one or the other, the *Address* is incremented while *Size* and *Required Size* are decremented. This simulates removing the available data from the front of the request. Then, *Add To Read Buffer* will search the cache and *Block Buffer* for the data at the end of the request. Every time a byte is found, the *Size* of the request is decremented by one. If the byte was a required byte then the *Required Size* is also decremented. This simulates removing any data available from the end of the request. *Add To Read Buffer* is either left with a request that has a *Size* equal to zero or the end points are both needed from memory. If the *Required Size* is zero then the request is a buffer hit, otherwise the request is a buffer miss. If the request is already a cache hit then the buffer hit is for some block management request. These kinds of buffer hits are not recorded because it would confuse the *Results Display* by making it possible to get a hit rate greater than 100%. If the *Size* is not zero and *Remove Read Duplicates* is equal to *No* then the request is added to the end of the *Read Buffer* using *Append*. *Append* is a buffer utility that adds the request to the end of the buffer. The request must be added to the end of the buffer in order not to interfere with *Memory Model*, which may be in the middle of a memory read. If *Remove Read*

Duplicates is equal to *Yes* then the first byte in the request will be spliced into the *Read Buffer*.

Splice is another buffer utility. *Splice* will first search the *Read Buffer* for the byte. If it can't find a request in the buffer that contains the byte then it will search for a memory request that is getting data from the same block. If one is found then the request is modified to include the new read byte request. If no suitable request can be found then *Splice* will add a one byte request to the *Read Buffer*. The *Address* is then incremented while *Size* and *Required Size* are decremented. Then, the cache and *Block Buffer* are searched for the next byte. If it is not found then the next byte is spliced into the *Read Buffer*. This process is repeated until all of the bytes of the request have either been spliced into the *Read Buffer* or found.

The *Buffer Hit* is normally defined as when the data is available but not in the cache. However, in order to support the testing of SACS, the definition of a buffer hit is revised to mean that a request was found to have accrued recently, and that given time to complete all block management requested data would have been in the cache. This allows *Test SACS* to predict the hits of a test run without taking into account the time it takes to perform the block management.

Every time a request is spliced into the read or write buffers, the *Time To Execute* and *Completion Time Estimate* must be recalculated. The new time estimates are performed by *Calculate Time Estimates*.

11. Search Cache

Search Cache is called by *Add To Read Buffer* to find any parts of the request that may already be located in the cache. This must be done because if a read request follows a write request using a write allocate policy, then part of the read may be in the cache while the rest may still need to be read from memory. *Search Cache* checks all *Cache Block Addresses* in the cache set. If any of the cache block addresses equal the block address of the byte, then *Search Cache* checks the *Cache Valid Bit* for the sub block that the byte is located in. If the sub block is valid then *Search Cache* returns *Yes*.

12. Add To Write Buffer

Add To Write Buffer adds one record to the write buffer. It also updates the *Read Buffer* if the *Update Read Buffer* argument is asserted. The process of updating the *Read Buffer* is simply changing the requests so that data made available by the write request is not requested from memory. *Update Read Buffer* should not be used unless the word and sub block sizes are equal. This is because a write request may reduce a read request to where the read request will not be large enough to validate a sub block. The write request may also be unable to set any valid bits because of sub block alignment. The result is that a sub block that was suppose to be read in is not.

D. MEMORY MODEL

Memory Model makes all the necessary calls to simulate main memory. *Memory Model* decides which calls to make based on *Memory Waiting For*. This function is called every time *Time* is incremented. If there are no read or write requests waiting to be

completed, the function does nothing. *Memory Model* contains a loop that forces the procedure to continue modeling until *TOA* and *TOD* are not equal to *Time*. This insures that if there are any events that occur in zero clock cycles then the next event is allowed to start.

Memory Model calls *Select Memory Request* to choose a request from either the read or the write buffers. *Memory Model* calls *Start Reads* and *Start Writes* to simulate accessing memory and receiving the first word of a memory request. *Continue Reads* and *Continue Writes* are then called to simulate the memory transfer of the following words of data.

1. Select Memory Request

Select Memory Request is called when memory is waiting for nothing. *Select Memory Request* chooses a request from either the read or write buffers based on priority. The request is not returned however, and the request is left at the top of the buffer with its *Priority* set to zeros and *Access In Progress* set equal to *Yes*. If a request is found, then *Memory Waiting For* is set to *Memory Read Request* or *Memory Write Request*, depending on whether the request was found in the read or write buffer.

2. Start Reads

Start Reads begins a read request, simulating the first word read from memory. The time to complete this read is called *Memory Access Time*. The *Block Buffer* is initialized in preparation to receive the new data words. If *Block Waiting For* is not equal to *Nothing* then *Start Reads* will have to wait before allowing the new memory read

request to start. If *Start Reads* does have to wait for the cache then *Memory Waiting For* is set equal to *Cache Update*, otherwise *Memory Waiting For* is set to *Memory Read Access*. The new block record is equal to the *Read Buffer* with its sizes set to zero. This gives the *Block Memory Request* the same block number as the *Read Memory Request*. The *Address* is aligned to *Word Size*. The *Address* must be aligned because the words read in will be aligned to Word Size. The new *Block Memory Request* is simply pushed onto the *Block Buffer*. *Block Waiting For* is set equal to *Memory Block Transfer* to indicate that data is being transferred from memory to the *Block Buffer*.

3. Continue Reads

Continue Memory Reads continues the memory read request started by *Start Memory Reads*. It simulates every read from memory other than the first word, which is simulated by *Start Memory Reads*. The time to complete each word transfer is equal to *Memory Transfer Time*. The block and read buffers are altered every time a word is read from memory. Once a request is complete, it is removed from the *Read Buffer* and *Memory Waiting For* is reset to *Nothing*. *Block Waiting For* is set to *Block Cache Access* in preparation to transfer the new data to the cache. If the *Completion Time Estimate* for the memory read request is not equal to Time then a time prediction error is raised.

4. Start Memory Writes

Start Memory Writes begins a memory write request, simulating the first word written to memory. The time to complete this one word write is called *Memory Access Time*. *Memory Waiting For* is set to *Memory Write Access*.

5. Continue Memory Writes

Continue Memory Writes continues the memory write request started by *Start Memory Writes*. Like *Continue Memory Reads*, it simulates every write to memory other than the first word, which is simulated by *Start Memory Writes*. The time to complete each word transfer is equal to *Memory Transfer Time*. The *Write Buffer* is altered every time a word is written to memory. Once the memory write request is complete, it is removed from the *Write Buffer*, and *Memory Waiting For* is reset to *Nothing*. If the *Completion Time Estimate* for the memory read request is not equal to *Time* when the request is completed then a time prediction error is raised.

6. Update Cache

Update Cache simulates entering data form the *Block Buffer* into the cache. *Update Cache* first checks whether or not the cache is busy. If it is not, then *Cache Busy* is asserted and *Block Waiting For* is set equal to *Block Cache Transfer*. The *Block TOA* is calculated to enable *Calculate Time Estimates* to predict the completion times for additional memory read requests in the buffer. If the cache is busy then the previous memory request time completions may be wrong. That is because all of the last estimates counted on the old *Block TOA*. Therefore they all must be recalculated.

Once the *Buffer Cache Access Time* has expired then *Block Waiting For* is set equal to *Nothing* and the *Cache Busy* is deasserted. The read data must then be removed from the *Block Buffer*. The appropriate sub blocks in the cache will then have their dirty bits cleared and valid bits set.

7. Add A Word To Memory Request

Add A Word To Memory Request adds a word to a *Memory Request* as if it had been read in from memory. The address is first aligned to *Word Size*. Then, the size is incremented by *Word Size*. This simulates the data being added to the request.

8. Remove A Word From Memory Request

Remove A Word Form Memory Request removes a word from a *Memory Request* as if it had been written to memory. A copy of the *Address* is first stored in *Old Address*. Then, the *Address* is word aligned and incremented by *Word Size*. The *Required Size* and *Size* are then decremented by the difference of the new *Address* and the *Old Address*. Finally, if the *Address* is outside the range of the original block, then *Address* is decremented by *Block Size* to simulate modulo addition This simulates removing a word from the memory request, taking into account word and block alignment constraints.

E. TIME ESTIMATES

Time estimates are performed to provide a method of testing Cache Model, and Memory Model's handling of the read and write buffers. These two procedures are located in "TimeEst.c"

1. Update Time To Execute

Update Time To Execute calculates the time to complete a memory transfer given *Memory Request*. *Memory Request* could be a read or write request in a buffer. *Update Time To Execute* changes the *Time To Execute* field to the new value. *Time To Execute* is calculated by first finding the number of *Words To Be Transferred*. If the

Memory Request is not being accessed then the *Time To Execute* is simply the *Access Time* plus the *Transfer Time* multiplied by one less than *Words To Be Written*. If the *Memory Request* is in progress then the new *Time To Execute* is dependent on *TOA* or *TOD* of the next word. *Memory Waiting For* dictates whether to use the *TOA* or *TOD*. If *Memory Waiting For* is equal to *Cache Update* then the request has not actually begun transferring data. Therefore, the *Time To Execute* can be calculated as if the read request is not in progress.

2. Calculate Time Estimates

Calculate Time Estimates updates the *Completion Time Estimates* for each request in both the read and write buffers. This function is called whenever the *Cache Model* adds to the read or write buffers. *Calculate Time Estimates* must be called every time new data is entered into the buffers. This is because all previous estimates did not take into account the new data requested. *Calculate Time Estimates* first orders all entries in both the *Read Buffer* and the *Write Buffer* by priority. Then, *Calculate Time Estimates* steps though both buffers simultaneously, each time picking the request that has the highest priority and adding the time to execute to the *Time Estimate*. The *Time Estimate* becomes that requests *Completion Time Estimate*. This process is repeated until all requests have a new *Completion Time Estimate*. *Time To Execute*, for cache request, is updated before it is used to calculate the *Time Estimate*.

VI. PROGRAM VALIDATION

A. TESTING SACS

Program validation was considered to be a paramount issue in designing and implementing SACS. The debugging techniques for SACS were engineered during the early planning phases, before any code was written. In fact, there was a great deal of energy spent trying to make SACS a general event simulator. Not only would this have made it easier for the user to alter the protocols, but more importantly, it would have been easier to test the program. It would have been easier because the number of different kinds of event transitions would have been less than the number of different cache argument permutations. This method was aborted because data that was stored in the cache and the buffers was completely different. Another problem that plagued this method was that the scoreboarding techniques were unique for each buffer. Having abandoned the previous method, and recognizing that SACS would have dozens of input parameters (37 to date), a great deal of concern developed over how the program could be tested. It was decided that hand testing would prove to be ineffective in eliminating most or all of the programming errors. Therefore, an automated testing routine was developed. The testing routine was incorporated into the source code of SACS and can be activated using the *-test* argument. When the program is in the test mode, it goes into an infinite loop generating pseudo-random load and store instructions. Each trace is processed using the same code as

if the trace was generated by a designer. Each time a new trace run is executed, the input parameters are randomized. This testing method is the backbone of all other validation methods for SACS. Other error checking is performed during this process. However, the random trace and random argument testing is the best method to ensure that all lines of code in SACS get executed during the test phase of SACS. This prevents SACS from experiencing a catastrophic failure during an actual simulation because almost all instructions are executed during the testing phase. SACS tries to predict the number of read and write hits for each run. These predictions are compared to the number of cache and buffer hits if the input arguments are expected to make the cache and buffer hits reflect the predictions. An example of when the input arguments would make it impossible to predict a hit or a miss is when the rand block replacement policy is used. A block of data that has just been read into the cache may be selected as a victim. This makes it impossible to predict which blocks will be in the cache at the beginning of the next request. Another example is when a read forward policy is used and the block buffer is not searched. Data expected to be in the cache may be in the *Block Buffer*, or in the *Read Buffer*, waiting to complete the block transfer. This requires that the predicted hits only be compared when the block and read buffers are searched. It also requires that during the test mode, buffer hits will include the read buffer. Policies that can not be checked for predicted hits are allowed in the test because they can be checked for other things including the simple, yet important requirement that the program does not spontaneously abort.

The instructions that are randomly generated for a test trace are seeded from 64 test vectors. These test vectors each have 7 read or write instructions. The number of these vectors to be used to generate a trace is randomly chosen. The actual test vectors to be used are also randomly chosen. Each test vector is assigned a random set of block addresses. Each of the block addresses for a particular test vector will map to the same set in the cache. The actual data address for each request is formed by taking the block address and adding a random number such that the data address is still in the same block. The size is also chosen randomly in such a way that the request does not violate block alignment.

Once all the test instructions have been created, they are randomly shuffled in such a way that the final number of hits and misses will remain the same as predicted.

B. CHECKING COMPLETION TIME

The timing analysis of SACS is so important that it was decided that every timing test that could be performed would be performed. SACS simulates events based on the current time only because some events can be predicted, such as the time that a buffer request would be removed from a buffer. A good timing test would be to calculate the estimated time of completion. Then, check to see if that estimated time is the same as the time that the request is removed from the buffer. If they do not match then an error is indicated. This kind of error checking goes on during every run whether it's a test run or a user's run.

C. CHECKING GLOBAL VARIABLES

SACS uses 82 global variables. Approximately half of the global variables are constant during a single run. These variables mostly represent input parameters, and although they are changed in between test runs, they remain constant for the rest of the trace run. SACS was written in C, a powerful language that permits the programmer to create some powerful and elusive bugs. Specifically, C allows assignments to be buried in logical expressions. This capability could easily result in altering input parameters instead of checking a parameters value. To avoid this kind of error and others like it, copies of all constant global variables are made before the beginning of the trace run. At the end of every simulated clock cycle these variables are compared to their original copies. If a discrepancy is found then an error is indicated.

Global variables that are not constant are also inspected. They are inspected to ensure that they are all within acceptable boundaries. These boundaries are not always constant. For example, histogram index and total time should always exceed Time.

VII. SAMPLE RUNS

A. EXAMPLE SACS SIMULATION RUN

In the first simulation run for SACS the default parameters were used. This run will demonstrate a write allocation miss and a write through hit. This run will also demonstrate a read miss that takes advantage of removing duplicates, and how a write request can update the read buffer. Table 4 shows the trace data used for the simulation run.

Table 4 TRACE DATA FOR RUN

Request Type	Request Address	Size	Time Until Next Request
w	x00000100	4	1
w	x00000108	8	1
r	x00000104	2	1

w indicates a write request
r indicates a read request

The *Request Type* is either a read or a write request. Read requests are indicated with a lower case "r". Write requests are indicated with a lower case "w". The address is read as a long hexadecimal integer. The *Request Size* is a long unsigned decimal integer. It represents the size in bytes. *Time Until Next Request* is the time between when the CPU's current request is complete and when the CPU makes the next request. The simulation run was performed by loading the trace data from Table 4 into an ASCII data

file named "SACS.Dat". Then, SACS was started with the trace mode on. The first trace display that SACS produced is shown in Figure 6.

```
Current Request:    Write                          Time:                1
Address:            00000100                       Next Request Time:   2
Size:               04
Cache Waiting For:  Write Cache Request
Memory Waiting For: Nothing                        Cache Hit:           Unknown
Block Waiting For:  Nothing                        Buffer Hit:          Unknown

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00000  00000000 0 0  0 0  0 0  0 0
      00000  00000000 0 0  0 0  0 0  0 0
      00000  00000000 0 0  0 0  0 0  0 0
      00000  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Write Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Block Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 6 Trace Display For Time=1

Figure 6 shows the status of SACS after the first clock cycle. *The Cache Waiting For* field shows that SACS is modeling the cache access. *Memory Waiting For* is *Nothing* because the cache does not know if the request is a hit or a miss. *Block Waiting For* is *Nothing* because no read requests have started. The *Next Request Time* indicates that the CPU will send another request at *Time* equal to 2. *Cache Hit* and *Buffer Hit* are *Unknown* because the cache block still has not been accessed. The request will be mapped to set number 16, block 64.

The trace display for *Time* equal to 2 is shown in Figure 7. This display shows that the CPU has made the second write request. Again, the *Cache Waiting For* indicates that the cache is accessing block 64. The *Memory Waiting For* indicates that memory is accessing the first word in memory for the last write request. The *TOD* indicates that the first word will be written to memory at *Time* equal to 5. Cache and buffer hits are again *Unknown* because block 64 has not been accessed. The set data indicates that the last write request at 100 validated the first sub block. The *Read Buffer* has the remaining data needed for block 64. The *Req.* field is the number of bytes required to satisfy the CPU request. None are required in this case because the data was only needed to fulfill a block management requirement. The *Write Buffer* shows the last write request. It also predicts that the memory write request will be complete at *Time* equal to 5.

```
Current Request:     Write                          Time:                2
Address:             00000108                       Next Request Time:   3
Size:                08                             TOD:                 5
Cache Waiting For:   Write Cache Request
Memory Waiting For:  Memory Write Access            Cache Hit:           Unknown
Block Waiting For:   Nothing                        Buffer Hit:          Unknown

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00064  00000100 1 0  0 0  0 0  0 0
      00065  00000000 0 0  0 0  0 0  0 0
      00066  00000000 0 0  0 0  0 0  0 0
      00067  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000104    12     00    00064      12          5           10

Write Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000100    04     00    00000      00          3            5

Block Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 7 Trace Display For Time=2

The next display is for *Time* equal to 3, and is shown in Figure 8. This display shows that cache is still working on the last write request. *Cache Waiting For* still indicates *Write Cache Request* because the default for a write hit requires one clock cycle after the cache has been accessed to update the cache. The memory is still waiting for the memory access of the first write memory request. The *Next Request Time* indicates that the CPU is waiting for the cache to finish with the last write request.

```
Current Request:     Write                          Time:                3
Address:             00000108                       Next Request Time:   3
Size:                08                             TOD:                 5
Cache Waiting For:   Write Cache Request
Memory Waiting For:  Memory Write Access            Cache Hit:           Yes
Block Waiting For:   Nothing                        Buffer Hit:          No

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00064  00000100 1 0  0 0  0 0  0 0
      00065  00000000 0 0  0 0  0 0  0 0
      00066  00000000 0 0  0 0  0 0  0 0
      00067  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000104    12     00     00064     12          5           10

Write Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000100    04     00     00000     00          3           5

Block Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 8 Trace Display For Time=3

Figure 9 shows the display for *Time* is equal to 4. The last write request has completed and the data provided by the write request was placed in block 64. As a result, the last two sub blocks are valid. The data was also placed into the *Write Buffer*. At first, it seems as though the two write memory requests should have been combined. The

resulting request would have had an address of 108 and a size of 12. Because words are accessed in memory in modulo form, the data for the first sub block would have been accessed last. However, because the last memory write request was in progress, the request could not modify its starting address. Therefore, two different requests resulted, and the *Read Buffer's* memory request size has been reduced by 8 bytes. This is an example of updating the Read Buffer with write data. This scoreboarding policy is a default for SACS, and can be disabled. The result is that the read request does not have to access data that was provided by a write request.

```
Current Request:     Read                          Time:                4
Address:             00000104                      Next Request Time:   6
Size:                02                            TOD:                 5
Cache Waiting For:   Read Cache Request
Memory Waiting For:  Memory Write Access           Cache Hit:           Unknown
Block Waiting For:   Nothing                       Buffer Hit:          Unknown

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00064  00000100 1 0  0 0  1 0  1 0
      00065  00000000 0 0  0 0  0 0  0 0
      00066  00000000 0 0  0 0  0 0  0 0
      00067  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address     Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000104     04     00    00064      12          3           12

Write Buffer Address     Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000100     04     00    00000      00          1            5
             00000108     08     00    00000      11          4            9

Block Buffer Address     Size   Req.   Block   Priority   Time Req.   Comp. Time

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 9 Trace Display For Time=4

Figure 10 shows the trace display for Time equal to 5. The last read request has turned out to be a miss. The buffers were also no help. At first glance, it might seem as though the read buffer should have provided a buffer hit. This would have been true if SACS were in a test mode. However, in a non test mode, the read buffer is not searched because the data is not really available. It still has to be read in from memory. The first write memory request starting at address 100 has completed, as predicted at *Time* equal to 5. The next memory request was chosen to be the write request because it had a high priority of 11, versus the read memory request with a priority of 12. Unfortunately, the request began before the cache could raise the priority to 10. Once the write request began, the priority went to 0 to prevent interruption.

```
Current Request:     Read                          Time:                5
Address:             00000104                      Next Request Time:   7
Size:                02                            TOD:                 8
Cache Waiting For:   Read Memory Request
Memory Waiting For:  Memory Write Access           Cache Hit:           No
Block Waiting For:   Nothing                       Buffer Hit:          No

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00064  00000100 1 0  0 0  1 0  1 0
      00065  00000000 0 0  0 0  0 0  0 0
      00066  00000000 0 0  0 0  0 0  0 0
      00067  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000104    04     02     00064     10          3          12

Write Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000108    08     00     00000     00          4           9

Block Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 10 Trace Display For Run #1, Time=5

From the *TOD* in Figure 10, it is obvious that little will happen until at least the first word is sent to memory. The display trace for *Time* equal to 8, when the first word is written to memory, is shown in Figure 11. It shows how SACS adjusts its buffers to represent individual words sent or rec. ed from memory. The *Memory Waiting For* switched to *Memory Write Transfer*, illustrating the transition from memory access to memory transfer. The *TOD* shows that the write will be complete at *Time* equal to 9, which is a lot better than the 3 clock cycles normally used to access memory.

```
Current Request:     Read                          Time:                 8
Address:             00000104                      Next Request Time:    10
Size:                02                            TOD:                  9
Cache Waiting For:   Read Memory Request
Memory Waiting For:  Memory Write Transfer         Cache Hit:            No
Block Waiting For:   Nothing                       Buffer Hit:           No

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00064  00000100 1 0  0 0  1 0  1 0
      00065  00000000 0 0  0 0  0 0  0 0
      00066  00000000 0 0  0 0  0 0  0 0
      00067  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000104     04     02   00064         10           3           12

Write Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             0000010C     04     00   00000         00           4            9

Block Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 11 Trace Display For Time=8

Figure 12 shows the completion of the write memory request, and the beginning of the read memory request at Time equal to 9. The Block Buffer has also been prepared to receive the read memory data. The Block Waiting For indicates that the Block Buffer is busy receiving read data from memory. The TOA shows that Time 12 is when the read memory request will be complete.

```
Current Request:     Read                          Time:                 9
Address:             00000104                      Next Request Time:    11
Size:                02                            TOD:                  12
Cache Waiting For:   Read Memory Request
Memory Waiting For:  Memory Read Access            Cache Hit:            No
Block Waiting For:   Memory Block Transfer         Buffer Hit:           No

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00064  00000100 1 0  0 0  1 0  1 0
      00065  00000000 0 0  0 0  0 0  0 0
      00066  00000000 0 0  0 0  0 0  0 0
      00067  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000104     04     02   00064         10           3           12

Write Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Block Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time
             00000104     00     00   00064         00           3           12

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 12 Trace Display For Time=9

Figure 13 shows the completion of the read memory request at Time equal to 12. The Cache Waiting For indicates Nothing because the request was satisfied when the required bytes arrived in the block buffer. The memory has nothing to do because both the read and the write buffers are empty. The simulation is not finished however, because the block buffer still has to update the cache with the new block data. *Block Waiting For* shows that the transfer is in progress.

```
Last Request:        Read                          Time:                12
Address:             00000104                      Next Request Time:   13
Size:                02
Cache Waiting For:   Nothing
Memory Waiting For:  Nothing                       Cache Hit:           No
Block Waiting For:   Block Cache Transfer          Buffer Hit:          No

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00064  00000100 1 0  0 0  1 0  1 0
      00065  00000000 0 0  0 0  0 0  0 0
      00066  00000000 0 0  0 0  0 0  0 0
      00067  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.    Comp. Time

Write Buffer Address    Size   Req.   Block   Priority   Time Req.    Comp. Time

Block Buffer Address    Size   Req.   Block   Priority   Time Req.    Comp. Time
             00000104    04     00    00064     00          1           13

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 13 Trace Display For Time=12

Figure 14 shows the trace display for Time equal to 13. This display shows how the block buffer updated block 64. One word of data was provided at address 104, making the second sub block valid. Once the block transfer was completed, SACS removed the memory request from the block buffer.

```
Last Request:        Read                          Time:                13
Address:             00000104                      Next Request Time:   13
Size:                02
Cache Waiting For:   Nothing
Memory Waiting For:  Nothing                       Cache Hit:           Unknown
Block Waiting For:   Nothing                       Buffer Hit:          Unknown

  Set Block  Address  V/D  V/D  V/D  V/D
00016 00064  00000100 1 0  1 0  1 0  1 0
      00065  00000000 0 0  0 0  0 0  0 0
      00066  00000000 0 0  0 0  0 0  0 0
      00067  00000000 0 0  0 0  0 0  0 0

Read Buffer  Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Write Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Block Buffer Address    Size   Req.   Block   Priority   Time Req.   Comp. Time

Next Command Please [ T, R, S, C, G #, #, -#, Help] >>>
```

Figure 14 Trace Display For Time=13

Figure 15 shows the results display. This display is always shown at the end of the run. It is interesting to note that even with the SACS default parameters, which lean toward providing the fastest response time to all CPU requests, the average access time for the one read miss was 8 clock cycles. Had this simulation been done with Dinero III, the user could only have assumed that a 2 byte read would have taken the memory access time of 3 clock cycles.

Requests Break Down

Request Types	Number of Requests	Number of Cache Hits	Number of Buffer Hits	Hit Rates	Miss Rates
Read	1	0	0	0.00%	100.00%
Write	2	1	0	50.00%	50.00%
Total	3	1	0	33.33%	66.67%

Request Time Histogram.

	Time=00	Time=01	Time=02	Time=03	Time>=04	Total	Ave Access Time
Read	0	0	0	0	1	8	8.000000
Write	0	1	1	0	0	3	1.500000
Ideal	2	0	1	0	0	2	

Next Command Please [T, R, S, C, G #, #, -#, Help] >>>□

Figure 15 Results Display

VIII. CONCLUSION

For more than a decade, caches have been designed and built. Despite the time and effort spent on cache designs, there seems to be no one design that has emerged as the best cache design. Even the most basic choices, such as associativity, block size, and whether to use a unified cache or two separate data and instruction caches, has not been a clear choice, or at least the correct choice was not agreed upon by all concerned.

The diversity of cache designs has been caused by budget constraints, changing memory technology, and changing CPU bandwidth requirements. Without proper timing information, matching the correct cache to the architecture is more of an art than a science. SACS offers a powerful tool in the early planning phase of a cache design. Its large set of scoreboarding, block management, and cache memory arguments allow the designer to survey different designs quickly. SACS is well documented and provides the designer with a number of debugging tools, including self-testing and global variable bounds checking. This makes modifying SACS to simulate a unique design feature extremely easy compared to other programs.

As mentioned throughout this paper, the most critical aspect of SACS is its ability to provide the designer with the average access time. Since the ultimate purpose of the cache is to minimize the average access time, any simulator that does not provide this number can only hope to provide the designer with superficial and misleading data.

Future developments of SACS will include more elaborate timing information. The number of histograms will expand to include what the CPU, memory, and block buffer were waiting for during a run. A new stall histogram will be introduced that will allow the user to easily modify SACS to analyze any combination of conditions. For example, how many times does a read request wait for access to the cache memory while an old write miss request updates the allocated block. A new global variable will allow the user to change all the histogram displays to probability density tables.

LIST OF REFERENCES

1. Patterson, D. A., and Hennessy, J. L., *Computer Organization & Design The Hardware/Software Interface*, p. 458-481, Morgan Kaufmann Publishers, Inc., 1994.

2 Jouppi, N. P., "Cache Write Policies and Performance," *Proceedings of the 20th Annual International Symposium on Computer Architecture*, p. 191-201, May 1993.

3. Smith, A. J., "Cache Memories," *Computing Surveys*, vol. 14-3, p. 473-530, September 1982.

4. Hennessy, J. L., and Patterson, D. A., *Computer Architecture, A Quantitative Approach*, p. 403-425, Morgan Kaufmann Publishers, Inc., 1990.

5. Hill, M. D., and others, "Wisconsin Architectural Research Tool Set," *Computer Architecture News*, v. 21-4, p. 8-10, September 1993.

APPENDIX

SOURCE CODE FOR SACS

```
/*****************************************************************************
**                                                          Page  1- 0      **
**                                SACS.h                                    **
**                                                                          **
**                          Part Of SACS 1.0                                **
**                    (StillAnother Cache Simulator)                        **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                          Page  1- 1   **
**                                SACS.h                                  **
**                                                                        **
**                     Still Another Cache Simulator                      **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       SACS.h defines all enumeration types.  It Contains forward       **
**  declarations of all functions used in SACS, (not just SACS.c). SACS.h **
**  also includes a list of all inline functions (macros).                **
**                                                                        **
**                            Table of Contents                           **
**                                                                        **
**          Cover Page ............................... Page  1- 1        **
**          Enumeration Definitions .................. Page  1- 2        **
**          Type Definitions ......................... Page  1- 4        **
**                                                                        **
**          Inline Function Definitions                                   **
**             SubBlock(Address) ...................... Page  1- 5        **
**             Set(Address) ........................... Page  1- 5        **
**             BlockAddress(Address) .................. Page  1- 5        **
**             WordAddress(Address) ................... Page  1- 5        **
**             SubBlockAddress(Address) ............... Page  1- 5        **
**                                                                        **
**          Complete List of Function Declarations within SACS            **
**             SACS.c ................................. Page  1- 6        **
**             Cache.c ................................ Page  1- 7        **
**             Memory.c ............................... Page  1- 8        **
**             TimeEst.c .............................. Page  1- 9        **
**             Get.c .................................. Page  1-10        **
**             Display.c .............................. Page  1-11        **
**             Record.c ............................... Page  1-12        **
**             Buffer.c ............................... Page  1-13        **
**             Array.c ................................ Page  1-14        **
**             TestingSACS.c .......................... Page  1-15        **
**             Checking.c ............................. Page  1-16        **
**                                                                        **
*****************************************************************************/

#ifndef __CACHE.H

#define __CACHE.H

#define ClearScreen "ClearScr"
```

```
/*****************************************************************************
**                                                          Page  1- 2  **
**                                SACS.h                                  **
**                                                                        **
**                        Enumeration Definitions                         **
**  Description:                                                          **
**                                                                        **
**       Listed below are the enumerations used in the SACS environment.  **
**  WaitingForTypes, and MemoryWaitingForTypes are on listed on the       **
**  following page.                                                       **
**                                                                        **
*****************************************************************************/


enum YesNoTypes                    {
                                   No,
                                   Yes,
                                   Unknown
                                   };

enum RequestTypes                  {
                                   None,
                                   Read,
                                   Write,
                                   NumberOfRequestsAvailable
                                   };

enum BlockReplacementPolicyTypes   {
                                   LRU,
                                   FIFO,
                                   RAND,
                                   NumberOfReplacementPoliciesAvailable
                                   };

enum WritePolicyTypes              {
                                   WriteThrough,
                                   WriteBack,
                                   NumberOfWritePoliciesAvailable
                                   };

enum WriteMissPolicyTypes          {
                                   WriteAround,
                                   WriteAllocate,
                                   NumberOfWriteMissPoliciesAvailable
                                   };
```

```
/*****************************************************************************
**                                                          Page  1- 3   **
**                                  SACS.h                                **
**                                                                        **
**                         Enumeration Definitions                        **
**                                continued                               **
**                                                                        **
*****************************************************************************/

enum CacheWaitingForTypes          {
                                   Nothing,
                                   CacheWaitingForReadCacheRequest,
                                   CacheWaitingForWriteCacheRequest,
                                   CacheWaitingForReadMemoryRequest,
                                   CacheWaitingForWriteMemoryRequest,
                                   CacheWaitingForFullReadBuffer,
                                   CacheWaitingForFullWriteBuffer,
                                   CacheWaitingForCPUCacheAccess,
                                   NumberOfCacheWaitingForsAvailable
                                   };

enum MemoryWaitingForTypes         {
                                   NothingTwo,
                                   MemoryWaitingForMemoryReadRequest,
                                   MemoryWaitingForMemoryReadAccess,
                                   MemoryWaitingForMemoryReadTransfer,
                                   MemoryWaitingForMemoryWriteRequest,
                                   MemoryWaitingForMemoryWriteAccess,
                                   MemoryWaitingForMemoryWriteTransfer,
                                   MemoryWaitingForCacheUpdate,
                                   NumberOfMemoryWaitingForsAvailable
                                   };

enum BlockWaitingForTypes          {
                                   NothingThree,
                                   MemoryBlockTransfer,
                                   BlockCacheAccess,
                                   BlockCacheTransfer,
                                   NumberOfBlockWaitingForsAvailable
                                   };
```

```
/*****************************************************************************
**                                                              Page  1- 4  **
**                                  SACS.h                                  **
**                                                                          **
**                             Type Definitions                             **
**  Description:                                                            **
**                                                                          **
**       These are all of the type definitions used in the SACS             **
**  environment, excluding enumeration types which are listed on the last   **
**  two pages.                                                              **
**                                                                          **
*****************************************************************************/


typedef unsigned long int TimeType;
typedef unsigned long int ScoreType;
typedef unsigned long int AddressType;
typedef unsigned long int CacheSizeType;
typedef unsigned      int SizeType;
typedef unsigned      int BufferSizeType;
typedef unsigned      int PriorityType;
typedef unsigned      int AssociativityType;
typedef unsigned      int HistogramIndexType;


typedef enum YesNoTypes                   YesNoType;
typedef enum RequestTypes                 RequestType;
typedef enum BlockReplacementPolicyTypes BlockReplacementPolicyType;
typedef enum WriteMissPolicyTypes         WriteMissPolicyType;
typedef enum WritePolicyTypes             WritePolicyType;
typedef enum CacheWaitingForTypes         CacheWaitingForType;
typedef enum MemoryWaitingForTypes        MemoryWaitingForType;
typedef enum BlockWaitingForTypes         BlockWaitingForType;


struct MemoryRequestStructType
   {
   AddressType   Address;
   SizeType      Size;
   SizeType      RequiredSize;
   SizeType      Block;
   PriorityType  Priority;
   YesNoType     AccessInProgress;
   TimeType      TimeToExecute;
   TimeType      CompletionTimeEstimate;
   };
typedef struct MemoryRequestStructType MemoryRequestType;


struct BufferStructType
   {
   MemoryRequestType   MemoryRequest[10];
   YesNoType           Full;
   YesNoType           Empty;
   BufferSizeType      Next;
   BufferSizeType      Max;
   CacheWaitingForType WaitingForFlag;
   };
typedef struct BufferStructType BufferType;
```

```
/*****************************************************************************
**                                                              Page  1- 5  **
**                                  SACS.h                                  **
**                                                                          **
**                       Inline Function Definitions                        **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       These macros act as inline functions. They are the only            **
**  macros which act as inline functions within the SACS environment,       **
**  except those located in "TestSACS.c".                                   **
**                                                                          **
*****************************************************************************/


#define SubBlock(Address)         (((Address)%BlockSize)/SubBlockSize)
#define Set(Address)              (((Address)/BlockSize)%NumberOfSets)

#define BlockAddress(Address)     (((Address)/BlockSize)*BlockSize)
#define WordAddress(Address)      (((Address)/WordSize)*WordSize)
#define SubBlockAddress(Address)  (((Address)/SubBlockSize)*SubBlockSize)
```

```
/*****************************************************************************
**                                                                Page  1- 6  **
**                                SACS.h                                      **
**                                                                            **
**                  List of SACS.c Function Declarations                      **
**                                                                            **
**  Description:                                                              **
**                                                                            **
**       This is a list of function declarations within the file scope        **
**  of "SACS.c".                                                              **
**                                                                            **
*****************************************************************************/


extern int                main();                                /* Page  2- 8 */
extern void                  LoadArguments();                     /* Page  2-11 */
extern unsigned long int       ScanArgument();                   /* Page  2-14 */
extern void                  InitializeProgrammersGlobalVariables();
                                                                  /* Page  2-15 */
extern void                  InitializeBuffers();                 /* Page  2-16 */
extern void                  DefineArrays();                      /* Page  2-17 */
extern void                  FreeArrays();                        /* Page  2-18 */
extern void                  OpenDataFile();                      /* Page  2-19 */
extern void                  CloseDataFile();                     /* Page  2-20 */
extern void                  PauseForCommand();                   /* Page  2-21 */
extern void                  Pause();                             /* Page  2-23 */
```

```
/*****************************************************************************
**                                                              Page  1- 7  **
**                                  SACS.h                                  **
**                                                                          **
**                  List of Cache.c Function Declarations                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "Cache.c".                                                           **
**                                                                          **
*****************************************************************************/


extern void      CacheModel();                              /* Page  4- 3 */
extern void         IsRequestAHit();                        /* Page  4- 4 */
extern void         ReadHit();                              /* Page  4- 5 */
extern void         ReadMiss();                             /* Page  4- 6 */
extern void         WriteHit();                             /* Page  4- 7 */
extern void         WriteMiss();                            /* Page  4- 8 */
extern void         AccessCache();                          /* Page  4-10 */

extern void      SelectBlockVictim();                       /* Page  4-11 */
extern void      SetDirtyBits();                            /* Page  4-13 */
extern void      WriteDirtySubBlocks();                     /* Page  4-14 */
extern void      AddToReadBuffer();                         /* Page  4-16 */
extern YesNoType    SearchCache();                          /* Page  4-20 */
extern void      AddToWriteBuffer();                        /* Page  4-21 */
```

```
/*****************************************************************************
**                                                              Page  1- 8  **
**                                  SACS.h                                  **
**                                                                          **
**                 List of Memory.c Function Declarations                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of Memory.c                                                             **
**                                                                          **
*****************************************************************************/

extern void MemoryModel();                                  /* Page  5- 3 */
extern void    SelectMemoryRequest();                       /* Page  5- 4 */
extern void    StartMemoryReads();                          /* Page  5- 5 */
extern void    ContinueMemoryReads();                       /* Page  5- 6 */
extern void    StartMemoryWrites();                         /* Page  5- 8 */
extern void    ContinueMemoryWrites();                      /* Page  5- 9 */
extern void UpdateCache();                                  /* Page  5-11 */
extern void AddAWordToMemoryRequest();                      /* Page  5-13 */
extern void RemoveAWordFromMemoryRequest();                 /* Page  5-14 */
```

```
/*****************************************************************************
**                                                              Page  1- 9  **
**                                  SACS.h                                  **
**                                                                          **
**                  List of TimeEst.c Function Declarations                 **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of TimeEst.c                                                            **
**                                                                          **
*****************************************************************************/


extern void UpdateTimeToExecute();                          /* Page  6- 3 */
extern void CalculateTimeEstimates();                       /* Page  6- 5 */
```

```
/*****************************************************************************
**                                                              Page  1-10  **
**                                  SACS.h                                  **
**                                                                          **
**                   List of Get.c Function Declarations                    **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "Get.c".                                                             **
**                                                                          **
*****************************************************************************/


extern void GetNextRequest();                                  /* Page  7- 3 */
extern void    GetNextFileRequest();                           /* Page  7- 5 */
extern void    GetNextKeyBoardRequest();                       /* Page  7- 6 */
```

```
/*****************************************************************************
**                                                              Page  1-11  **
**                                  SACS.h                                  **
**                                                                          **
**                 List of Display.c Function Declarations                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "Display.c".                                                         **
**                                                                          **
*****************************************************************************/


extern void      DisplayTrace();                          /* Page  8- 3 */
extern void         DisplayCurrentRequest();              /* Page  8- 4 */
extern void         DisplayWaitingFors();                 /* Page  8- 5 */
extern void         DisplayBlock();                       /* Page  8- 6 */
extern void         DisplayBuffers();                     /* Page  8- 7 */
extern void            DisplayBuffer();                   /* Page  8- 8 */

extern void      DisplayRequestsBreakDown();              /* Page  8- 9 */
extern void         DisplayRequestHistogram();            /* Page  8-11 */

extern void      DisplayStallHistogram();                 /* Page  8-13 */
extern ScoreType    LastScreenHistogramScore();           /* Page  8-14 */
extern void      DisplayCacheArguments();                 /* Page  8-15 */
extern void      DisplayHelp();                           /* Page  8-17 */

extern void      DisplayTestingHeader();                  /* Page  8-18 */

extern void      PrintYesNo();                            /* Page  8-19 */
extern void      PrintRequest();                          /* Page  8-19 */
extern void      PrintReplacementPolicy();                /* Page  8-19 */
extern void      PrintWritePolicy();                      /* Page  8-19 */
extern void      PrintWriteMissPolicy();                  /* Page  8-19 */
extern void      PrintWaitingFor();                       /* Page  8-19 */
extern void      PrintMemoryWaitingFor();                 /* Page  8-19 */
extern void      PrintBlockWaitingFor();                  /* Page  8-19 */

extern void      PrintTime();                             /* Page  8-20 */
extern void      PrintTimeCentered();                     /* Page  8-20 */
extern void      PrintScoreCentered();                    /* Page  8-20 */
extern void      PrintAddress();                          /* Page  8-20 */
extern void      PrintCacheSize();                        /* Page  8-20 */
extern void      PrintSize();                             /* Page  8-20 */
extern void      PrintSize2();                            /* Page  8-20 */
extern void      PrintBufferSize();                       /* Page  8-21 */
extern void      PrintPriority();                         /* Page  8-21 */
extern void      PrintAssociativity();                    /* Page  8-21 */
extern void      PrintHistogramIndex();                   /* Page  8-21 */

extern void      PrintBit();                              /* Page  8-22 */
extern void      PrintPercent();                          /* Page  8-22 */
extern void      PrintAveAccess();                        /* Page  8-22 */
```

```
/*****************************************************************************
**                                                              Page  1-12 **
**                                  SACS.h                                  **
**                                                                          **
**                  List of Record.c Function Declarations                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "Record.c".                                                          **
**                                                                          **
*****************************************************************************/


extern void  RecordRequest();                                  /* Page  9- 3 */
extern void  RecordStall();                                    /* Page  9- 5 */
extern void  RecordForMatlab();                                /* Page  9- 7 */
```

```
/*****************************************************************************
**                                                                Page  1-13  **
**                                  SACS.h                                   **
**                                                                           **
**                  List of Buffer.c Function Declarations                   **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       This is a list of functions declarations within the file scope      **
**  of "Buffer.c".                                                           **
**                                                                           **
*****************************************************************************/


extern void              Push();                          /* Page 10- 3 */
extern MemoryRequestType Pop();                           /* Page 10- 4 */
extern void              ChangeTopMemoryRequest();        /* Page 10- 5 */
extern void              Append();                        /* Page 10- 6 */
extern MemoryRequestType View();                          /* Page 10- 7 */
extern void              Clear();                         /* Page 10- 8 */
extern void              Order();                         /* Page 10- 9 */
extern void              Splice();                        /* Page 10-10 */
extern YesNoType         Search();                        /* Page 10-12 */
extern YesNoType         UpdatingReadBuffer();            /* Page 10-13 */
extern void              RemoveZeroSizes();               /* Page 10-15 */
extern YesNoType         NoRequestsLeft();                /* Page 10-16 */
```

```
/*****************************************************************************
**                                                                Page  1-14  **
**                                  SACS.h                                   **
**                                                                           **
**                  List of Array.c Function Declarations                    **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       This is a list of function declarations within the file scope       **
**  of "Array.c".                                                            **
**                                                                           **
*****************************************************************************/


extern int  *DefineArray1D();                                  /* Page 11- 3 */
extern int **DefineArray2D();                                  /* Page 11- 4 */
extern void  FreeArray1D();                                    /* Page 11- 5 */
extern void  FreeArray2D();                                    /* Page 11- 6 */
```

```
/*****************************************************************************
**                                                              Page  1-15  **
**                                  SACS.h                                  **
**                                                                          **
**                List of TestSACS.c Function Declarations                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of functions declarations within the file scope     **
**  of "TestSACS.c".                                                        **
**                                                                          **
*****************************************************************************/


extern void      ChangeArguments();                          /* Page 12- 6 */
extern void      TestSACS();                                 /* Page 12- 8 */
extern void         CreateInstructionSets();                 /* Page 12- 9 */
extern void         ShufflingInstructionSets();              /* Page 12-12 */
extern YesNoType       CanBeSwitched();                      /* Page 12-14 */
extern void         WriteInstructionSet();                   /* Page 12-15 */
```

```
/*****************************************************************************
**                                                              Page  1-16  **
**                                  SACS.h                                  **
**                                                                          **
**                  List of Checking.c Function Declarations                **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "Checking.c".                                                        **
**                                                                          **
*****************************************************************************/


extern void Checking();                                    /* Page 13- 3 */
extern void    CheckingConstants();                        /* Page 13- 4 */
extern void       PrintConstError();                       /* Page 13-11 */
extern void    CheckingForValuesOutOfBounds();             /* Page 13-12 */
extern void       PrintTimeBoundaryError();                /* Page 13-15 */
extern void       PrintScoreBoundaryError();               /* Page 13-16 */
extern void       PrintSizeBoundaryError();                /* Page 13-17 */
extern void       PrintEnumBoundaryError();                /* Page 13-18 */
extern void    CheckingForInconsistencies();               /* Page 13-19 */
extern void       PrintTotalTimeError();                   /* Page 13-21 */
extern void       PrintTotalScoreError();                  /* Page 13-22 */
extern void CheckingPredictions();                         /* Page 13-23 */
extern void    PrintScorePredictionError();                /* Page 13-24 */
extern void PrintTimePredictionError();                    /* Page 13-25 */


#endif
```

```
/*****************************************************************************
**                                                          Page  2- 0   **
**                                SACS.c                                  **
**                                                                        **
**                          Part Of SACS 1.0                              **
**                    (StillAnother Cache Simulator)                      **
**                                                                        **
**   Program Modified: 3/17/94                                            **
**   File Modified:    3/17/94                                            **
**                                                                        **
**   Author:  William G. Smith                                            **
**   Address:  Electrical Engineering Department                          **
**             Naval Postgraduate School                                  **
**             Monterey, CA 93940                                         **
**                                                                        **
**   Copyright 1994, William G. Smith                                     **
**                                                                        **
**   Permission to use, copy, modify, and distribute this software and    **
**   its documentation for any purpose and without fee is hereby granted  **
**   provided that the above copyright notice appears in all copies.  No  **
**   modified version of this program should be redistributed without the **
**   authors consent.  William G. Smith makes no warranty or              **
**   representation, promise of guarantee, either expressed or implied,   **
**   with respect to this software's ability to produce valid results.    **
**   This program is provided "as is" any financial, personal or property **
**   damage caused by the use of this program is the responsibility of the **
**   user.                                                                **
**                                                                        **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                                Page  2- 1  **
**                                SACS.c                                     **
**                                                                           **
**                    Still Another Cache Simulator                          **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       SACS simulates all functions one clock cycle at a time using a      **
**  global variable named Time.  Normally it is preferred to preform timing **
**  simulations using event queues so that time can advance to the next     **
**  event.  However, in most cache simulations so many things happen in one **
**  clock cycle that en event queue would probably not improve the          **
**  preformance of the simulator                                             **
**                                                                           **
**       In the main event loop of SACS, Time is incremented one clock       **
**  cycle at a time.  Time is never changed by any other procedure.          **
**  The requests are entered into the simulation from GetNextRequest.        **
**  Simulation of all events is performed by the Main Event Loop calling     **
**  CacheModel, MemoryModel, and UpdateCache.                                **
**                                                                           **
**       The main procedure of SACS seems to call simulations in a fairly    **
**  strange order.  This is because SACS is insuring that all events that    **
**  can be started, during a particular clock cycle are started, and that    **
**  all events that can bomplete during a particular clock cycle do.  It     **
**  also gives an inherent priority to the cache access events.              **
**  Specifically accesses from the CPU to the cache are given higher         **
**  priority that accesses from the BlockBuffer.  This is why the Update     **
**  Cache procedure is found in three different places in the main loop.     **
**  Memory Model calls are found before and after the CacheModel.  This      **
**  allows memory events that are to complete during a clock cycle to do     **
**  so.  The Cache Model will then have the benefit of the newly arrived     **
**  data.  The MemoryModel call after the CacheModel call insures that any   **
**  new memory request made by the cache are started that clock cycle.       **
**                                                                           **
**       SACS's main loop includes the source code to control testing,       **
**  checking, and tracing.  The DesiredTime variable is controlled entirely  **
**  by the MainEventLoop.  DesiredTime represents a user request to advance  **
**  the simulation to a particular time without the trace on.  SACS can not  **
**  run Time backwards.  However if the Desired Time.  The user can make     **
**  time requests using arguments "G #", "#", "-#".                          **
**                                                                           **
**       Throughout MainEventLoop, CacheWaitingFor is checked to see if      **
**  it's equal to Nothing.  This indicates that the last request has been    **
**  serviced and that the cache is ready for the next request.  The          **
**  procedures that model specific events as ReadHit, ReadMiss, and          **
**  AccessCache are called repeatedly during their simulations they use      **
**  Cache Waiting For and Time to determine what to do next.  If any of      **
**  these procedures needs to wait for a period either to simulate an        **
**  access or because a resource is not available, then they will set Cache  **
**  Waiting For to the appropriate value.  The modeling procedures in        **
**  Memory Model work the same way using Memory Waiting For.                 **
**                                                                           **
**       Whenever SACS finds an error or a discrepancy then the boolean      **
**  variable Discrepancy Found is set to Yes.  This forces SACS into a       **
**  trace mode so that the user may try to identify the cause of the error.  **
**  In test mode a discrepancy forces SACS out of test mode so that the      **
**  trace file that caused the error is not erased by a new file.            **
**                                                                           **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                          Page  2- 2      **
**                                SACS.c                                    **
**                                                                          **
**                      Still Another Cache Simulator                       **
**                               continued                                  **
**                                                                          **
**                                                                          **
**      SACS.c contains the source code for main(), which contains the      **
**  main loop.  All initialization of global variables, array definitions,  **
**  and file management are done inside "SACS.c".                           **
**                                                                          **
**      For information on what SACS does see the User's Guide.             **
**                                                                          **
**      For information on how to run SACS see the User's Guide.            **
**                                                                          **
**      For information on how to modify SACS see the Programmer's Guide.   **
**                                                                          **
**      For information on how SACS works see the Programmer's Guide.       **
**                                                                          **
**                            Table of Contents                             **
**                                                                          **
**         Cover Page ............................... Page  2- 1           **
**         User Defined Global Variables ............ Page  2- 3           **
**         Programmer Defined Global Variables ....... Page  2- 4           **
**            Enumerator Strings ..................... Page  2- 5           **
**         List of SACS.c Function Declarations ...... Page  2- 7           **
**         main() ................................... Page  2- 8           **
**            LoadArguments() ........................ Page  2-11           **
**               ScanArgument() ...................... Page  2-14           **
**            InitializeProgrammersGlobalVariables() . Page  2-15           **
**            InitializeBuffers() .................... Page  2-16           **
**            DefineArrays() ......................... Page  2-17           **
**            FreeArrays() ........................... Page  2-18           **
**            OpenDataFile() ......................... Page  2-19           **
**            CloseDataFile() ........................ Page  2-20           **
**            PauseForCommand() ...................... Page  2-21           **
**            Pause() ................................ Page  2-23           **
**                                                                          **
*****************************************************************************/


#include <stdlib.h>
#include <stdio.h>

#include "SACS.h"
```

```
/*****************************************************************************
**                                                          Page  2- 3  **
**                                SACS.c                                  **
**                                                                        **
**                      User Defined Global Variables                     **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       These variables represent the programs input parameters.         **
**                                                                        **
*****************************************************************************/


CacheSizeType              CacheSize                = 8192;          /* -cs   */
SizeType                   BlockSize                =   16;          /* -bs   */
SizeType                   SubBlockSize             =    4;          /* -sbs  */
AssociativityType          Associativity            =    4;          /* -a    */
SizeType                   WordSize                 =    4;          /* -ws   */

TimeType                   ReadCacheAccessTime      =    1;          /* -rcat */
TimeType                   ReadCacheHitTime         =    0;          /* -rcht */
TimeType                   ReadCacheMissTime        =    0;          /* -rcmt */
TimeType                   WriteCacheAccessTime     =    1;          /* -wcat */
TimeType                   WriteCacheHitTime        =    1;          /* -wcht */
TimeType                   WriteCacheMissTime       =    0;          /* -wcmt */

TimeType                   MemoryAccessTime         =    3;          /* -mat  */
TimeType                   MemoryTransferTime       =    1;          /* -mtt  */
TimeType                   BufferCacheAccessTime    =    1;          /* -bcat */

BufferSizeType             ReadBufferSize           =    4;          /* -rbs  */
BufferSizeType             WriteBufferSize          =    4;          /* -wbs  */

BlockReplacementPolicyType BlockReplacementPolicy   = LRU;           /* -brp  */
WritePolicyType            WritePolicy              = WriteThrough; /* -wp   */
WriteMissPolicyType        WriteMissPolicy          = WriteAllocate;/* -wmp  */

YesNoType          ReadForward                  = Yes;    /* -rf    -drf    */
YesNoType          CPUWaitsForCacheWrites       = No;     /* -cwfcw -dcwfcw */
YesNoType          SearchBlockBuffer            = Yes;    /* -sbb   -dsbb   */
YesNoType          UpdateReadBuffer             = Yes;    /* -urb   -durb   */
YesNoType          RemoveReadDuplicates         = Yes;    /* -rrd   -drrd   */
YesNoType          RemoveWriteDuplicates        = Yes;    /* -rwd   -drwd   */

PriorityType       ReadPriority                 =   1;        /* -rpr       */
PriorityType       WritePriority                =   2;        /* -wpr       */
PriorityType       ReadForWriteAllocatePriority =   3;        /* -rfwapr    */
PriorityType       WriteDirtyBlockPriority      =   4;        /* -wdbpr     */
PriorityType       NoPriority                   = 100;        /* -npr       */

YesNoType          Trace                        =  No;        /* -t    -dt  */
YesNoType          Check                        = Yes;        /* -c    -dc  */
YesNoType          Test                         =  No;        /* -test      */

YesNoType          KeyBoardIO                   =  No;        /* -kbio -fio */
char               *DataFileName                = "SACS.Dat"; /* -f         */

HistogramIndexType ScreenHistogramMaxIndex      =   5;        /* -shmi      */
HistogramIndexType FileHistogramMaxIndex        =  10;        /* -fhmi      */
```

```
/*****************************************************************************
**                                                          Page  2- 4  **
**                                SACS.c                                  **
**                                                                        **
**                   Programmer Defined Global Variables.                 **
**                                                                        **
*****************************************************************************/

TimeType              Time;
TimeType              DesiredTime;
CacheWaitingForType   CacheWaitingFor;
MemoryWaitingForType  MemoryWaitingFor;
BlockWaitingForType   BlockWaitingFor;
YesNoType             DiscrepancyFound;

YesNoType             CacheHit;
YesNoType             BufferHit;
YesNoType             CacheBusy;

RequestType           Request;
RequestType           LastRequest;
AddressType           RequestAddress;
SizeType              RequestSize;
SizeType              RequestBlockNumber;
TimeType              TimeOfNextRequest;

SizeType              NumberOfBlocks;
SizeType              NumberOfSubBlocks;
SizeType              NumberOfSets;

AddressType           *CacheBlockAddress;     /* [NumberOfBlocks]            */
TimeType              *LastCacheBlockAccessTime;
SizeType              *CacheNextBlock;        /* [NumberOfSets]              */
YesNoType             **CacheValidBit;        /* [NumberOfBlocks]            */
YesNoType             **CacheDirtyBit;        /* [NumberOfSubBlocks]         */

TimeType              **RequestTimeHistogram; /* [NumberOfRequestsAvailable] */
                                              /* [FileHistogramMaxIndex]     */
TimeType              **StallTimeHistogram; /* [NumberOfWaitingForsAvailable]*/
                                            /* [FileHistogramMaxIndex]       */
TimeType              *TotalRequestTime;    /* [NumberOfRequestsAvailable]   */
TimeType              *TotalStallTime;      /* [NumberOfWaitingForsAvailable]*/
ScoreType             *NumberOfAccesses;    /* [NumberOfRequestsAvailable]   */
ScoreType             *NumberOfCacheHits;
ScoreType             *NumberOfBufferHits;
ScoreType             *PredictedNumberOfAccesses;
ScoreType             *PredictedNumberOfHits;
ScoreType             TotalNumberOfAccesses   = 0;  /* Not reset during test */

ScoreType             TotalNumberOfWordsReadFromMemory;
ScoreType             TotalNumberOfWordsWrittenToMemory;
ScoreType             TotalNumberOfWordsWrittenToCache;

BufferType            ReadBuffer;
BufferType            WriteBuffer;
BufferType            BlockBuffer;

AddressType           MAR;
TimeType              TOA;
TimeType              TOD;
TimeType              BlockTOA;

FILE                  *DataFile;
YesNoType             EndOfDataFile;
```

```
/*****************************************************************************
**                                                                Page 2- 5  **
**                                  SACS.c                                  **
**                                                                          **
**                    Programmer Defined Global Variables                   **
**                                 continued                                **
**                                                                          **
**                            Enumerator Strings                            **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Enumerator strings are string copies of enumeration types.  These  **
**  are used for display purposes.                                          **
**                                                                          **
*****************************************************************************/


char *YesNoString[3]=
      {
      "No     ",
      "Yes    ",
      "Unknown"
      };

char *RequestString[NumberOfRequestsAvailable]=
      {
      "None ",
      "Read ",
      "Write"
      };

char *ReplacementPolicyString[NumberOfReplacementPoliciesAvailable] =
      {
      "LRU ",
      "FIFO",
      "RAND"
      };

char *WritePolicyString[NumberOfWritePoliciesAvailable] =
      {
      "Write Though",
      "Write Back  "
      };

char *WriteMissPolicyString[NumberOfWriteMissPoliciesAvailable] =
      {
      "Write Around  ",
      "Write Allocate"
      };
```

```
/*****************************************************************************
**                                                                Page 2- 6  **
**                                  SACS.c                                  **
**                                                                          **
**                    Programmer Defined Global Variables                   **
**                                 continued                                **
**                                                                          **
**                            Enumerator Strings                            **
**                                 continued                                **
**  Description:                                                            **
**                                                                          **
**        Enumerator strings are string copies of enumeration types.  These **
**  are used for display purposes.                                          **
**                                                                          **
*****************************************************************************/


char *CacheWaitingForString[NumberOfCacheWaitingForsAvailable]=
      {
      "Nothing              ",
      "Read Cache Request   ",
      "Write Cache Request  ",
      "Read Memory Request  ",
      "Write Memory Request",
      "Full Read Buffer     ",
      "Full Write Buffer    ",
      "CPU Cache Access     "
      };

char *MemoryWaitingForString[NumberOfMemoryWaitingForsAvailable]=
      {
      "Nothing               ",
      "Memory Read Request   ",
      "Memory Read Access    ",
      "Memory Read Transfer  ",
      "Memory Write Request  ",
      "Memory Write Access   ",
      "Memory Write Transfer",
      "Cache Update          "
      };

char *BlockWaitingForString[NumberOfBlockWaitingForsAvailable]=
      {
      "Nothing               ",
      "Memory Block Transfer",
      "Block Cache Access    ",
      "Block Cache Transfer  "
      };
```

```
/*****************************************************************************
**                                                              Page  2- 7  **
**                                  SACS.c                                  **
**                                                                          **
**                   List of SACS.c Function Declarations                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "SACS.c".                                                            **
**                                                                          **
*****************************************************************************/


int                main();                                  /* Page  2- 8 */
void                  LoadArguments();                      /* Page  2-11 */
unsigned long int        ScanArgument();                    /* Page  2-14 */
void                  InitializeProgrammersGlobalVariables(); /* Page  2-15 */
void                  InitializeBuffers();                  /* Page  2-16 */
void                  DefineArrays();                       /* Page  2-17 */
void                  FreeArrays();                         /* Page  2-18 */
void                  OpenDataFile();                       /* Page  2-19 */
void                  CloseDataFile();                      /* Page  2-20 */
void                  PauseForCommand();                    /* Page  2-21 */
void                  Pause();                              /* Page  2-23 */
```

```
/*****************************************************************************
**                                                          Page  2- 8  **
**                                SACS.c                                  **
**                                                                        **
**                                 main                                   **
**                                                                        **
*****************************************************************************/


main(argc, argv)
   int argc;
   char *argv[];

   {

   LoadArguments(argc,argv);

   if (KeyBoardIO==No || Test==Yes) OpenDataFile();

   Time=0;

   while (Time==0 || Test==Yes)

      {

      if (Test==Yes) ChangeArguments();
      InitializeProgrammersGlobalVariables();
      InitializeBuffers();
      DefineArrays();

      if (Test==Yes) TestSACS(PredictedNumberOfAccesses, PredictedNumberOfHits);

      RecordRequest(NumberOfRequestsAvailable); /* Reseting LastTimes */
      RecordStall(NumberOfCacheWaitingForsAvailable);

      CheckingConstants(Yes);

      GetNextRequest();
      CacheHit=Unknown;
      BufferHit=Unknown;
```

```
/*****************************************************************************
**                                                          Page  2- 9  **
**                                SACS.c                                  **
**                                                                        **
**                           Main Event Loop.                             **
**                                                                        **
*****************************************************************************/

      while ((Request+CacheWaitingFor+MemoryWaitingFor>Nothing ||
              Time<=TimeOfNextRequest) &&
             DiscrepancyFound==No      &&
             Time>0                     )
         {

         if (BlockWaitingFor==BlockCacheTransfer) UpdateCache();
         MemoryModel();
         CacheModel();
         if (BlockWaitingFor==BlockCacheAccess && BufferCacheAccessTime==0)
            UpdateCache();
         MemoryModel();

         RecordRequest(Request);
         if (CacheWaitingFor==Nothing) Request=None;
         RecordRequest(Request);
         RecordStall(CacheWaitingFor);

         if (Time==DesiredTime) { Trace=Yes; DesiredTime=0; }

         if (CacheWaitingFor!=Nothing &&
            ((CPUWaitsForCacheWrites && Request==Write) || Request==Read))
            TimeOfNextRequest++;

         if (Time>=TimeOfNextRequest && CacheWaitingFor==Nothing)
            {
            GetNextRequest();
            CacheHit=Unknown; BufferHit=Unknown;
            if (Request==None)
               {
               if (BlockWaitingFor==BlockCacheAccess) UpdateCache();
               Time++;
               RecordStall(CacheWaitingFor);
               RecordRequest(Request);
               Time--;
               if (Check) Checking();
               if (Trace) PauseForCommand();
               Time++;
               }
            }
         else
            {
            if (BlockWaitingFor==BlockCacheAccess) UpdateCache();
            Time++;
            RecordStall(CacheWaitingFor);
            RecordRequest(Request);
            Time--;
            if (Check) Checking();
            if (Trace) PauseForCommand();
            Time++;
            }
         if (Time>DesiredTime && DesiredTime!=0) Time=0;
         }
```

```
/*****************************************************************************
**                                                                Page  2-10  **
**                                SACS.c                                     **
**                                                                           **
**                         End Of Main Event Loop.                           **
**                                                                           **
*****************************************************************************/


      if (Test==Yes && DiscrepancyFound==No)
         {

         CheckingPredictions();

         TotalNumberOfAccesses+=NumberOfAccesses[Read]
                               +NumberOfAccesses[Write];

         }

/*if (TotalNumberOfAccesses>=11270) {Trace=Yes; Test=No;}*/

      if (DiscrepancyFound==Yes)
         {
         Pause();
         Trace=Yes;
         Test=No;
         DesiredTime=0;
         Time=0;
         }

      if (DiscrepancyFound==No && Test==No && Time!=0)
         {
         DisplayRequestsBreakDown();
         RecordForMatlab();
         }

      FreeArrays();

      rewind(DataFile);
      EndOfDataFile=No;

      }

   if (KeyBoardIO==No) CloseDataFile();

   return(0);

   }
```

```
/*****************************************************************************
**                                                                Page  2-11  **
**                                  SACS.c                                  **
**                                                                          **
**                              LoadArguments                               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       LoadArguments takes the argument list argv and changes the         **
**  user defined global variables (See Page 2-3).                           **
**                                                                          **
*****************************************************************************/


void LoadArguments(argc,argv)

   int argc;
   char *argv[];

   {

   int i,j;

   for (i=1; i<argc; i++)
      {

      if (!(strcmp(argv[i],"-cs"   )))
         CacheSize                 = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-bs"   )))
         BlockSize                 = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-sbs"  )))
         SubBlockSize              = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-a"    )))
         Associativity             = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-ws"   )))
         WordSize                  = ScanArgument(argv[++i]);

      if (!(strcmp(argv[i],"-rcat" )))
         ReadCacheAccessTime       = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-rcht" )))
         ReadCacheHitTime          = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-rcmt" )))
         ReadCacheMissTime         = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-wcat" )))
         WriteCacheAccessTime      = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-wcht" )))
         WriteCacheHitTime         = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-wcmt" )))
         WriteCacheMissTime        = ScanArgument(argv[++i]);

      if (!(strcmp(argv[i],"-mat"  )))
         MemoryAccessTime          = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-mtt"  )))
         MemoryTransferTime        = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-bcat" )))
         BufferCacheAccessTime     = ScanArgument(argv[++i]);

      if (!(strcmp(argv[i],"-rbs"  )))
         ReadBufferSize            = ScanArgument(argv[++i]);
      if (!(strcmp(argv[i],"-wbs"  )))
         WriteBufferSize           = ScanArgument(argv[++i]);
```

```
      if (!(strcmp(argv[i],"-brp")))
         {
         BlockReplacementPolicy=-1;
         for (j=0; j<NumberOfReplacementPoliciesAvailable; j++)
            {
            if (!(strcmp(argv[i],ReplacementPolicyString[j])))
               BlockReplacementPolicy=j;
            }
         if (BlockReplacementPolicy<0)
            {
            printf("Invalid Block Replacement Policy");
            exit(1);
            }
         }

      if (!(strcmp(argv[i],"-wp")))
         {
         WritePolicy=-1;
         for (j=0; j<NumberOfWritePoliciesAvailable; j++)
            {
            if (!(strcmp(argv[i],WritePolicyString[j])))
               WritePolicy=j;
            }
         if (WritePolicy<0)
            {
            printf("Invalid Write Policy");
            exit(1);
            }
         }

      if (!(strcmp(argv[i],"-wmp")))
         {
         WriteMissPolicy=-1;
         for (j=0; j<NumberOfWriteMissPoliciesAvailable; j++)
            {
            if (!(strcmp(argv[i],WriteMissPolicyString[j])))
               WriteMissPolicy=j;
            }
         if (WriteMissPolicy<0)
            {
            printf("Invalid Write Miss Policy");
            exit(1);
            }
         }
```

```
    if (!(strcmp(argv[i],"-rf"    ))) ReadForward              = Yes;
    if (!(strcmp(argv[i],"-drf"   ))) ReadForward              = No;
    if (!(strcmp(argv[i],"-cwfcw" ))) CPUWaitsForCacheWrites   = Yes;
    if (!(strcmp(argv[i],"-dcwfcw"))) CPUWaitsForCacheWrites   = No;
    if (!(strcmp(argv[i],"-sbb"   ))) SearchBlockBuffer        = Yes;
    if (!(strcmp(argv[i],"-dsbb"  ))) SearchBlockBuffer        = No;
    if (!(strcmp(argv[i],"-urb"   ))) UpdateReadBuffer         = Yes;
    if (!(strcmp(argv[i],"-durb"  ))) UpdateReadBuffer         = No;
    if (!(strcmp(argv[i],"-rrd"   ))) RemoveReadDuplicates     = Yes;
    if (!(strcmp(argv[i],"-drrd"  ))) RemoveReadDuplicates     = No;
    if (!(strcmp(argv[i],"-rwd"   ))) RemoveWriteDuplicates    = Yes;
    if (!(strcmp(argv[i],"-drwd"  ))) RemoveWriteDuplicates    = No;

    if (!(strcmp(argv[i],"-rpr"   )))
       ReadPriority                  = ScanArgument(argv[++i]);
    if (!(strcmp(argv[i],"-wpr"   )))
       WritePriority                 = ScanArgument(argv[++i]);

    if (!(strcmp(argv[i],"-rfwapr")))
       ReadForWriteAllocatePriority = ScanArgument(argv[++i]);

    if (!(strcmp(argv[i],"-wdbpr" )))
       WriteDirtyBlockPriority       = ScanArgument(argv[++i]);

    if (!(strcmp(argv[i],"-npr"   )))
       WriteDirtyBlockPriority       = ScanArgument(argv[++i]);

    if (!(strcmp(argv[i],"-t"    ))) Trace                    = Yes;
    if (!(strcmp(argv[i],"-dt"   ))) Trace                    = No;
    if (!(strcmp(argv[i],"-c"    ))) Check                    = Yes;
    if (!(strcmp(argv[i],"-dc"   ))) Check                    = No;
    if (!(strcmp(argv[i],"-test" ))) Test                     = Yes;

    if (!(strcmp(argv[i],"-kbio" ))) KeyBoardIO               = Yes;
    if (!(strcmp(argv[i],"-fio"  ))) KeyBoardIO               = No;
    if (!(strcmp(argv[i],"-f"    ))) DataFileName             = argv[++i];

    if (!(strcmp(argv[i],"-shmi" )))
       ScreenHistogramMaxIndex       = ScanArgument(argv[++i]);
    if (!(strcmp(argv[i],"-fhmi" )))
       FileHistogramMaxIndex         = ScanArgument(argv[++i]);

    }

  }
```

```
/*****************************************************************************
**                                                                Page  2-14  **
**                                  SACS.c                                  **
**                                                                          **
**                               ScanArgument                               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       ScanArgument scans the input string for an unsigned long int,      **
**  if one is not found an error is raised.                                 **
**                                                                          **
*****************************************************************************/


unsigned long int ScanArgument(Argument)
   char *Argument;

   {

   unsigned long int Temp;

   if (sscanf(Argument,"%U",&Temp)!=1)
      {
      printf("Error unsigned integer expected [%s].",Argument);
      };

   return(Temp);

   }
```

```
/*****************************************************************************
**                                                          Page  2-15   **
**                                SACS.c                                  **
**                                                                        **
**                   InitializeProgrammersGlobalVariables                 **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       InitializeProgrammersGlobalVariables takes the user defined global **
**  variables and calculates programmer defined global variables, which are **
**  constant, once the input paramaters are determined, and reinitializes  **
**  the global variables what will change.                                **
**                                                                        **
*****************************************************************************/


void InitializeProgrammersGlobalVariables()

   {

   Time                              = 1;

   CacheWaitingFor                   = Nothing;
   MemoryWaitingFor                  = Nothing;
   BlockWaitingFor                   = Nothing;
   DiscrepancyFound                  = No;

   CacheHit                          = Unknown;
   BufferHit                         = Unknown;
   CacheBusy                         = No;

   Request                           = None;
   LastRequest                       = None;
   RequestAddress                    = 0;
   RequestBlockNumber                = 0;
   RequestSize                       = 0;
   RequestBlockNumber                = 0;
   TimeOfNextRequest                 = 0;

   NumberOfBlocks                    = CacheSize/BlockSize;
   NumberOfSubBlocks                 = BlockSize/SubBlockSize;
   NumberOfSets                      = NumberOfBlocks/Associativity;

   TotalNumberOfWordsReadFromMemory  = 0;
   TotalNumberOfWordsWrittenToMemory = 0;
   TotalNumberOfWordsWrittenToCache  = 0;

/*
   ReadsLeftForBlock                 = 0;
   ReadsLeftForRequest               = 0;
   WritesLeftForBlock                = 0;
   WritesLeftForRequest              = 0;
*/
   MAR                               = 0;
   TOA                               = 0;
   TOD                               = 0;
   BlockTOA                          = 0;

   EndOfDataFile                     = No;

   }
```

```
/*****************************************************************************
**                                                              Page  2-16  **
**                                  SACS.c                                  **
**                                                                          **
**                             InitializeBuffers                            **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       InitializeBuffers places the buffers in an empty state, with       **
**  their Max values set to the appropriate size.                           **
**                                                                          **
*****************************************************************************/


void InitializeBuffers()

   {

   ReadBuffer.Full  = No;
   ReadBuffer.Empty = Yes;
   ReadBuffer.Next  = 0;

   WriteBuffer      = ReadBuffer;
   BlockBuffer      = ReadBuffer;

   ReadBuffer.Max   = ReadBufferSize-1;
   WriteBuffer.Max  = WriteBufferSize-1;
   BlockBuffer.Max  = 0;

   ReadBuffer.WaitingForFlag  = CacheWaitingForFullReadBuffer;
   WriteBuffer.WaitingForFlag = CacheWaitingForFullWriteBuffer;
   BlockBuffer.WaitingForFlag = Nothing;

   }
```

```
/*****************************************************************************
**                                                                          **
**                                SACS.c                                    **
**                                                                          **
**                             DefineArrays                                 **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       DefineArrays assigns memory to the array pointers.                 **
**                                                                          **
*****************************************************************************/


void DefineArrays()

   {

   CacheBlockAddress           = (AddressType*)
                                 DefineArray1D(NumberOfBlocks,
                                               sizeof(AddressType));
   LastCacheBlockAccessTime    = (TimeType*)
                                 DefineArray1D(NumberOfBlocks,
                                               sizeof(TimeType));
   CacheNextBlock              = (SizeType*)
                                 DefineArray1D(NumberOfSets,
                                               sizeof(SizeType));
   CacheValidBit               = (YesNoType**)
                                 DefineArray2D(NumberOfBlocks,
                                               NumberOfSubBlocks,
                                               sizeof(YesNoType));
   CacheDirtyBit               = (YesNoType**)
                                 DefineArray2D(NumberOfBlocks,
                                               NumberOfSubBlocks,
                                               sizeof(YesNoType));
   RequestTimeHistogram        = (TimeType**)
                                 DefineArray2D(NumberOfRequestsAvailable,
                                               FileHistogramMaxIndex,
                                               sizeof(TimeType));
   StallTimeHistogram          = (TimeType**)
                                 DefineArray2D(NumberOfCacheWaitingForsAvailable,
                                               FileHistogramMaxIndex,
                                               sizeof(TimeType));
   TotalRequestTime            = (TimeType*)
                                 DefineArray1D(NumberOfRequestsAvailable,
                                               sizeof(TimeType));
   TotalStallTime              = (TimeType*)
                                 DefineArray1D(NumberOfCacheWaitingForsAvailable,
                                               sizeof(TimeType));
   NumberOfAccesses            = (ScoreType* )
                                 DefineArray1D(NumberOfRequestsAvailable,
                                               sizeof(ScoreType));
   NumberOfCacheHits           = (ScoreType* )
                                 DefineArray1D(NumberOfRequestsAvailable,
                                               sizeof(ScoreType));
   NumberOfBufferHits           = (ScoreType* )
                                 DefineArray1D(NumberOfRequestsAvailable,
                                               sizeof(ScoreType));
   PredictedNumberOfAccesses   = (ScoreType* )
                                 DefineArray1D(NumberOfRequestsAvailable,
                                               sizeof(ScoreType));
   PredictedNumberOfHits       = (ScoreType* )
                                 DefineArray1D(NumberOfRequestsAvailable,
                                               sizeof(ScoreType));

   }
```

```
/*****************************************************************************
**                                                          Page  2-18   **
**                                SACS.c                                  **
**                                                                        **
**                              FreeArrays                                **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       FreeArrays deallocates the memory assigned to the array points   **
**  by DefineArrays.                                                      **
**                                                                        **
*****************************************************************************/


void FreeArrays()

   {
char c;

   FreeArray1D(CacheBlockAddress,          NumberOfBlocks);
   FreeArray1D(LastCacheBlockAccessTime,   NumberOfBlocks);
   FreeArray1D(CacheNextBlock,             NumberOfSets);
   FreeArray2D(CacheValidBit,              NumberOfBlocks, NumberOfSubBlocks);
   FreeArray2D(CacheDirtyBit,              NumberOfBlocks, NumberOfSubBlocks);

   FreeArray2D(RequestTimeHistogram,       NumberOfRequestsAvailable,
                                           FileHistogramMaxIndex);
   FreeArray2D(StallTimeHistogram,         NumberOfCacheWaitingForsAvailable,
                                           FileHistogramMaxIndex);

   FreeArray1D(TotalRequestTime,           NumberOfRequestsAvailable);
   FreeArray1D(TotalStallTime,             NumberOfCacheWaitingForsAvailable);

   FreeArray1D(NumberOfAccesses,           NumberOfRequestsAvailable);
   FreeArray1D(NumberOfCacheHits,          NumberOfRequestsAvailable);
   FreeArray1D(NumberOfBufferHits,         NumberOfRequestsAvailable);
   FreeArray1D(PredictedNumberOfAccesses,  NumberOfRequestsAvailable);

   FreeArray1D(PredictedNumberOfHits,      NumberOfRequestsAvailable);

   }
```

```
/*****************************************************************************
**                                                                Page  2-19  **
**                                  SACS.c                                  **
**                                                                          **
**                               OpenDataFile                               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       OpenDataFile opens the file specified by DataFileName for          **
**  reading.  This becomes the data file that GetNextFileRequest reads      **
**  from.                                                                   **
**                                                                          **
*****************************************************************************/


void OpenDataFile()

   {

   if (Test==No)
      {
      if ((DataFile=fopen(DataFileName,"r"))==NULL)
         {
         printf("Cannot open %s file",DataFileName);
         exit(0);
         }
      }
   else
      {
      if ((DataFile=fopen(DataFileName,"w+"))==NULL)
         {
         printf("Cannot open %s file",DataFileName);
         exit(0);
         }
      }

   }
```

```
/*****************************************************************************
**                                                                Page  2-20  **
**                                  SACS.c                                  **
**                                                                          **
**                              CloseDataFile                               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       CloseDataFile closes the data file that OpenDataFile opened.       **
**                                                                          **
*****************************************************************************/

void CloseDataFile()

   {

   fclose(DataFile);

   }
```

```
/*****************************************************************************
**                                                                Page  2-21  **
**                                  SACS.c                                  **
**                                                                          **
**                              PauseForCommand                             **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       PauseForCommand controles the displays.  It takes input for the    **
**  keyboard to determan which display to provide.  It also adjusts the     **
**  global variable Desired Time based on "#", "-#", and "G #" commands.    **
**                                                                          **
*****************************************************************************/


void PauseForCommand()

   {

   static char LastDisplayMode=' ';

   char     InputString[255],
            *TmpStringPt,
            CommandChar,
            DisplayMode=' ';

   int      Index;
   TimeType TmpTime;

   if (Trace==Yes) LastDisplayMode='t'; else LastDisplayMode='r';

   while (DisplayMode!=LastDisplayMode)

      {

      if (DisplayMode!=' ') LastDisplayMode=DisplayMode;
      DisplayMode=LastDisplayMode;

      if (LastDisplayMode=='t') DisplayTrace();
      if (LastDisplayMode=='r') DisplayRequestsBreakDown();
      if (LastDisplayMode=='s') DisplayStallHistogram();
      if (LastDisplayMode=='c') DisplayCacheArguments();
      if (LastDisplayMode=='h') DisplayHelp();

      printf("\nNext Command Please [ T, R, S, C, G #, #, -#, Help] >>>");

      Index=-1;
      do
         {
         Index++;
         scanf("%c",&InputString[Index]);
         }
                        while(InputString[Index]!='\n');

      while (InputString[0]==' ')
         for (Index=0; InputString[Index]!='\n'; Index++)
            InputString[Index]=InputString[Index+1];

      CommandChar=InputString[0];

      if (CommandChar>='A' && CommandChar<='Z') CommandChar+=('a'-'A');
```

```
      if (sscanf(InputString,"%U",&TmpTime)==1 && CommandChar!='\n')
         {
         DesiredTime=Time+TmpTime;
         Trace=No;
         }

      if (CommandChar=='-')
         {
         TmpStringPt=InputString;
         TmpStringPt++;
         if (sscanf(TmpStringPt,"%U",&TmpTime)==1) DesiredTime=Time-TmpTime;
         Trace=No;
         }

      if (CommandChar=='g')
         {
         TmpStringPt=InputString;
         TmpStringPt++;
         if (sscanf(TmpStringPt,"%U",&TmpTime)==1) DesiredTime=TmpTime;
         Trace=No;
         }


      if (CommandChar=='t') DisplayMode='t';
      if (CommandChar=='r') DisplayMode='r';
      if (CommandChar=='s') DisplayMode='s';
      if (CommandChar=='c') DisplayMode='c';
      if (CommandChar=='h') DisplayMode='h';
      if (CommandChar=='q') exit(0);

      }

   }
```

```
/*****************************************************************************
**                                                              Page  2-23  **
**                                  SACS.c                                  **
**                                                                          **
**                                  Pause                                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Waiting for a character to be entered in.                          **
**                                                                          **
*****************************************************************************/

void Pause()

   {

   char InputCharacter;

   printf("\nHit the return key to Continue:");

   do
      {
      scanf("%c",&InputCharacter);
      }
   while (InputCharacter!='\n');

   }
```

```
/*****************************************************************************
**                                                          Page  3- 0      **
**                                Global.h                                  **
**                                                                          **
**                            Part Of SACS 1.0                              **
**                      (StillAnother Cache Simulator)                      **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                              Page  3- 1  **
**                                 Global.h                                 **
**                                                                          **
**                  Global Variables Used by SACS Packages                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Global.h is the only include file needed by all of the SACS source **
**  files.   It contains all the global variables, both user and programmer **
**  defined variables.  The user defined variables represent all the input  **
**  parameters.  The programmer defined variables represent all global      **
**  variables that are shared between all the SACS source code files, that  **
**  the user does not have access to.                                       **
**                                                                          **
**       SACS.c defines all of the initial values of the global variables   **
**  therefore, does not include Global.h                                    **
**                                                                          **
**                             Table of Contents                            **
**                                                                          **
**          Cover Page ............................... Page  3- 1          **
**          User Defined Global Variables ............ Page  3- 2          **
**          Programmer Defined Global Variables ....... Page  3- 3          **
**                                                                          **
*****************************************************************************/


#ifndef __GLOBAL.H
#define __GLOBAL.H

#include <stdlib.h>
#include <stdio.h>

#include "SACS.h"
```

```
/*****************************************************************************
**                                                          Page  3- 2   **
**                                Global.h                                **
**                                                                        **
**                     User Defined Global Variables                      **
**  Description:                                                          **
**                                                                        **
**         These variables represent the programs input parameters.       **
**                                                                        **
*****************************************************************************/


extern CacheSizeType              CacheSize;
extern SizeType                   BlockSize;
extern SizeType                   SubBlockSize;
extern AssociativityType          Associativity;
extern SizeType                   WordSize;

extern TimeType                   ReadCacheAccessTime;
extern TimeType                   ReadCacheHitTime;
extern TimeType                   ReadCacheMissTime;
extern TimeType                   WriteCacheAccessTime;
extern TimeType                   WriteCacheHitTime;
extern TimeType                   WriteCacheMissTime;

extern TimeType                   MemoryAccessTime;
extern TimeType                   MemoryTransferTime;
extern TimeType                   BufferCacheAccessTime;

extern BufferSizeType             ReadBufferSize;
extern BufferSizeType             WriteBufferSize;

extern BlockReplacementPolicyType BlockReplacementPolicy;
extern WritePolicyType            WritePolicy;
extern WriteMissPolicyType        WriteMissPolicy;

extern YesNoType                  ReadForward;
extern YesNoType                  CPUWaitsForCacheWrites;
extern YesNoType                  SearchBlockBuffer;
extern YesNoType                  UpdateReadBuffer;
extern YesNoType                  RemoveReadDuplicates;
extern YesNoType                  RemoveWriteDuplicates;

extern PriorityType               ReadPriority;
extern PriorityType               WritePriority;
extern PriorityType               ReadForWriteAllocatePriority;
extern PriorityType               WriteDirtyBlockPriority;
extern PriorityType               NoPriority;

extern YesNoType                  Trace;
extern YesNoType                  Check;
extern YesNoType                  Test;

extern YesNoType                  KeyBoardIO;
extern char                       *DataFileName;

extern HistogramIndexType         ScreenHistogramMaxIndex;
extern HistogramIndexType         FileHistogramMaxIndex;
```

```
/*****************************************************************************
**                                                          Page  3- 3  **
**                                Global.h                                **
**                                                                        **
**                   Programmer Defined Global Variables                  **
**                                                                        **
*****************************************************************************/

extern TimeType              Time;
extern TimeType              DesiredTime;
extern CacheWaitingForType   CacheWaitingFor;
extern MemoryWaitingForType  MemoryWaitingFor;
extern BlockWaitingForType   BlockWaitingFor;
extern YesNoType             DiscrepancyFound;

extern YesNoType     CacheHit;
extern YesNoType     BufferHit;
extern YesNoType     CacheBusy;

extern RequestType   Request;
extern RequestType   LastRequest;
extern AddressType   RequestAddress;
extern SizeType      RequestSize;
extern SizeType      RequestBlockNumber;
extern TimeType      TimeOfNextRequest;

extern SizeType      NumberOfBlocks;
extern SizeType      NumberOfSubBlocks;
extern SizeType      NumberOfSets;

extern AddressType   *CacheBlockAddress;     /* [NumberOfBlocks]            */
extern TimeType      *LastCacheBlockAccessTime;
extern SizeType      *CacheNextBlock;        /* [NumberOfSets]              */
extern YesNoType     **CacheValidBit;        /* [NumberOfBlocks]            */
extern YesNoType     **CacheDirtyBit;        /* [NumberOfSubBlocks]         */

extern TimeType      **RequestTimeHistogram; /* [NumberOfRequestsAvailable] */
                                             /* [FileHistgramMaxIndex]      */
extern TimeType      **StallTimeHistogram;   /* [NumberOfCacheWaitingForsAv]*/
                                             /* [FileHistgramMaxIndex]      */
extern TimeType      *TotalRequestTime;      /* [NumberOfRequestsAvailable] */
extern TimeType      *TotalStallTime;        /* [NumberOfStallsAvailable]   */
extern ScoreType     *NumberOfAccesses;      /* [NumberOfRequestsAvailable] */
extern ScoreType     *NumberOfCacheHits;
extern ScoreType     *NumberOfBufferHits;
extern ScoreType     *PredictedNumberOfAccesses;
extern ScoreType     *PredictedNumberOfHits;
extern ScoreType     TotalNumberOfAccesses;

extern ScoreType     TotalNumberOfWordsReadFromMemory;
extern ScoreType     TotalNumberOfWordsWrittenToMemory;
extern ScoreType     TotalNumberOfWordsWrittenToCache;

extern BufferType    ReadBuffer;
extern BufferType    WriteBuffer;
extern BufferType    BlockBuffer;

extern AddressType   MAR;
extern TimeType      TOA;
extern TimeType      TOD;
extern TimeType      BlockTOA;

extern FILE          *DataFile;
extern YesNoType     EndOfDataFile;
```

```
/*****************************************************************************
**                                                          Page  3- 4  **
**                                Global.h                                **
**                                                                        **
**                   Programer Defined Global Variables                   **
**                                continued                               **
**                            Enumerator Strings                          **
**                                                                        **
*****************************************************************************/


extern char

   *YesNoString[],                  /* [2]                                 */
   *RequestString[],                /* [NumberOfRequestsAvailable]         */
   *ReplacementPolicyString[],      /* [ReplacementPolicyString]           */
   *WritePolicyString[],            /* [NumberOfWritePoliciesAvailable]    */
   *WriteMissPolicyString[],        /* [NumberOfWriteMissPoliciesAvailable] */

   *CacheWaitingForString[],        /* [NumberOfCacheWaitingForsAvailable]  */
   *MemoryWaitingForString[],       /* [NumberOfMemoryWaitingForsAvailable] */
   *BlockWaitingForString[];        /* [NumberOfBlockWaitingForsAvailable]  */


#endif
```

```
/*****************************************************************************
**                                                          Page  4- 0      **
**                                Cache.c                                   **
**                                                                          **
**                            Part Of SACS 1.0                              **
**                      (StillAnother Cache Simulator)                      **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                          Page  4- 1      **
**                                Cache.c                                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       CacheModel makes all the necessary calls to simulate cache memory. **
**  CacheModel decides which calls to make, based on the value of CacheHit, **
**  and Request.  This function is called every time Time is incremented.  **
**  If there are no read or write requests waiting to be completed the     **
**  function does nothing.  The value of CacheHit will remain Unknown until **
**  the appropriate cache access time has expired.  Then CacheModel will   **
**  call IsRequestAHit to determine if the request is a hit or a miss.     **
**                                                                          **
**                             Table of Contents                            **
**                                                                          **
**          Cover Page ............................... Page  4- 1          **
**          List of Cache.c Function Declarations ..... Page  4- 2          **
**          CacheModel() ............................. Page  4- 3          **
**             IsRequestAHit() ....................... Page  4- 4          **
**             ReadHit() ............................. Page  4- 5          **
**             ReadMiss() ............................ Page  4- 6          **
**             WriteHit() ............................ Page  4- 7          **
**             WriteMiss() ........................... Page  4- 8          **
**             AccessCache() ......................... Page  4-10          **
**                                                                          **
**          SelectBlockVictim() ...................... Page  4-11          **
**          SetDirtyBits() ........................... Page  4-13          **
**          WriteDirtySubBlocks() .................... Page  4-14          **
**          AddToReadBuffer() ........................ Page  4-16          **
**             SearchCache() ......................... Page  4-20          **
**          AddToWriteBuffer() ....................... Page  4-21          **
**                                                                          **
*****************************************************************************/

#include "Global.h"
```

```
/*****************************************************************************
**                                                                Page  4- 2  **
**                                Cache.c                                    **
**                                                                           **
**                 List of Cache.c Function Declarations                     **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       This is a list of function declarations within the file scope       **
**  of "Cache.c".                                                            **
**                                                                           **
*****************************************************************************/


void      CacheModel();                                    /* Page  4- 3 */
void         IsRequestAHit();                              /* Page  4- 4 */
void         ReadHit();                                    /* Page  4- 5 */
void         ReadMiss();                                   /* Page  4- 6 */
void         WriteHit();                                   /* Page  4- 7 */
void         WriteMiss();                                  /* Page  4- 8 */
void         AccessCache();                                /* Page  4-10 */

void      SelectBlockVictim();                             /* Page  4-11 */
void      SetDirtyBits();                                  /* Page  4-13 */
void      WriteDirtySubBlocks();                           /* Page  4-14 */
void      AddToReadBuffer();                               /* Page  4-16 */
YesNoType    SearchCache();                                /* Page  4-20 */
void      AddToWriteBuffer();                              /* Page  4-21 */
```

```
/*****************************************************************************
**                                                          Page  4- 3   **
**                                Cache.c                                  **
**                                                                         **
**                               CacheModel                                **
**                                                                         **
**  Description:                                                           **
**                                                                         **
**       CacheModel makes all the necessary calls to simulate cache memory. **
**  CacheModel decides which calls to make, based on the value of CacheHit, **
**  and Request.  This function is called every time Time is incremented.   **
**  If there are no read or write requests waiting to be completed the      **
**  function does nothing.  The value of CacheHit will remain Unknown until **
**  the appropriate cache access time has expired.  Then CacheModel will    **
**  call IsRequestAHit to determine if the request is a hit or a miss.      **
**                                                                         **
*****************************************************************************/

void CacheModel()

   {

   if (CacheHit==Unknown && Request!=None)
      {
      if (Request==Read )
         AccessCache(ReadCacheAccessTime, CacheWaitingForReadCacheRequest);
      if (Request==Write)
         AccessCache(WriteCacheAccessTime,CacheWaitingForWriteCacheRequest);
      if (CacheWaitingFor==Nothing) IsRequestAHit();
      }

   if (CacheHit==Yes && Request==Read ) ReadHit();
   if (CacheHit==No  && Request==Read ) ReadMiss();
   if (CacheHit==Yes && Request==Write) WriteHit();
   if (CacheHit==No  && Request==Write) WriteMiss();

   }
```

```
/*****************************************************************************
**                                                          Page  4- 4   **
**                                 Cache.c                                 **
**                                                                         **
**                              IsRequestAHit                              **
**                                                                         **
**  Description:                                                           **
**                                                                         **
**       IsRequestAHit determines if the request is a hit or a miss, and   **
**  sets CacheHit to the appropriate value.  IsRequestAHit will find the   **
**  SetNumber that the data is supposed to be in.  Then all                **
**  CacheBlockAddresses in that set will be checked to see if they equal   **
**  the BlockAddress for that request.  If the correct block is found,     **
**  then all sub blocks that are required to satisfy the request will be   **
**  inspected for validity.  If all required sub blocks are valid then     **
**  CacheHit will equal Yes on return from IsRequestAHit.                  **
**                                                                         **
*****************************************************************************/


void IsRequestAHit ()

   {

   SizeType   SetNumber  = Set(RequestAddress);
   SizeType   FirstBlock = SetNumber*Associativity;
   SizeType   LastBlock  = FirstBlock+Associativity-1;
   SizeType   BlockIndex;
   SizeType   SubBlockIndex;

   CacheHit=No;
   BufferHit=Unknown;

   for (BlockIndex=FirstBlock; BlockIndex<=LastBlock; BlockIndex++)
      {

      if (CacheBlockAddress[BlockIndex]==BlockAddress(RequestAddress))
         {
         CacheHit=Yes;
         if (Request==Read)
            {
            for (SubBlockIndex=SubBlock(RequestAddress);
                 SubBlockIndex<=SubBlock(RequestAddress+RequestSize-1);
                 SubBlockIndex++)
               if (CacheValidBit[BlockIndex][SubBlockIndex]==No) CacheHit=No;
            }
         LastCacheBlockAccessTime[BlockIndex]=Time;
         }

      }

   if (CacheHit==Yes) BufferHit=No;

   }
```

```
/*****************************************************************************
**                                                          Page  4- 5   **
**                                Cache.c                                  **
**                                                                         **
**                                ReadHit                                  **
**                                                                         **
**  Description:                                                           **
**                                                                         **
**       ReadHit is called to simulate a cache hit during a read request.  **
**  Read Hit simply finishes simulating the cache access for the hit.      **
**  ReadCachHitTime is the time required to send the data from the cache   **
**  to the CPU.  Note that the Ttime to locate the block in the cache is   **
**  simulated in CacheModel.  ReadHit is called repeatedly while Time is   **
**  incremented until Access Cache returns with CacheWaitingFor equal to   **
**  Nothing.  AccessCache will return CacheWaitingFor equal to             **
**  ReadCacheRequest until the ReadCacheHitTime has expired.               **
**                                                                         **
*****************************************************************************/


void ReadHit()

   {

   AccessCache(ReadCacheHitTime,CacheWaitingForReadCacheRequest);

   }
```

```
/*****************************************************************************
**                                                          Page  4- 6  **
**                                 Cache.c                                **
**                                                                        **
**                                ReadMiss                                **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       ReadMiss is called to simulate a cache miss during a read request. **
**  ReadMiss first simulates the time it would take to perform all block   **
**  management for a read miss.  This time is called Read Cache Miss Time. **
**  Once that time has passed Read Miss calls Select Block Victim to pick a **
**  block in the set.  When SelectBlockVictim returns with CacheWaitingFor **
**  equal to Nothing the Request Block Number will contain the new block   **
**  number where the data will be placed.                                 **
**                                                                        **
**       Once the new block has been chosen, ReadMiss will call           **
**  AddToReadBuffer.  If ReadForward is selected, then RequiredSize for the **
**  memory request will be equal BlockSize.  The RequiredSize in the read  **
**  memory request tells the MemoryModel how much of the requested data    **
**  must be read into the BlockBuffer before resetting Cache WaitingFor    **
**  back to Nothing.  By setting RequiredSize equal to BlockSize, Read Miss **
**  is forcing Memory Model to read in the entire block before setting     **
**  Cache aiting For back to Nothing.  Once the Memory Model has read in   **
**  the data, it is assumed to be able to the CPU during that clock cycle. **
**                                                                        **
*****************************************************************************/

void ReadMiss()

   {

   AccessCache(ReadCacheMissTime,CacheWaitingForReadCacheRequest);

   if (CacheWaitingFor==Nothing ||
       CacheWaitingFor==CacheWaitingForFullWriteBuffer)
      SelectBlockVictim();

   if (CacheWaitingFor==Nothing ||
       CacheWaitingFor==CacheWaitingForFullReadBuffer)
      {
      if (ReadForward==Yes)
         AddToReadBuffer(RequestAddress,
                         BlockSize,
                         RequestSize,
                         RequestBlockNumber,
                         ReadPriority);
      else
         AddToReadBuffer(RequestAddress,
                         BlockSize,
                         BlockSize,
                         RequestBlockNumber,
                         ReadPriority);
      if (CacheWaitingFor==Nothing)
         CacheWaitingFor=CacheWaitingForReadMemoryRequest;
      }

   RecordStall(CacheWaitingFor);

   if (CacheWaitingFor==CacheWaitingForReadMemoryRequest &&
       NoRequestsLeft(&ReadBuffer))
      CacheWaitingFor=Nothing;

   }
```

```
/*****************************************************************************
**                                                          Page  4- 7   **
**                                Cache.c                                 **
**                                                                        **
**                                WriteHit                                **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       WriteHit is called to simulate a cache hit during a write request. **
**  Write Hit will first simulate the time to write the data to the       **
**  RequestBlockNumber in the cache.  Note that the time to locate the    **
**  block was simulated by CacheModel.  Once WriteCacheHitTime has expired **
**  then WriteHit will perform the block management for the request.  The **
**  block management is dictated by the WritePolicy.  For a WriteBack     **
**  policy the sub blocks written to must have their dirty bits set.  This **
**  is done by SetDirtyBit.  For a WriteThrough policy the request must be **
**  set to the write buffer.  This is done by AddToWriteBuffer.           **
**                                                                        **
*****************************************************************************/


void WriteHit()

   {

   AddressType TempAddress;

   AccessCache(WriteCacheHitTime,CacheWaitingForWriteCacheRequest);

   if (CacheWaitingFor==Nothing ||
       CacheWaitingFor==CacheWaitingForFullWriteBuffer)
      {
      switch (WritePolicy)
         {
         case WriteBack:
            {
            SetDirtyBits();
            break;
            }
         case WriteThrough:
            {
            for (TempAddress =SubBlockAddress(RequestAddress+SubBlockSize-1);
                 TempAddress <SubBlockAddress(RequestAddress+RequestSize);
                 TempAddress+=SubBlockSize)
               CacheValidBit[RequestBlockNumber][SubBlock(TempAddress)]=Yes;
            AddToWriteBuffer(RequestAddress,RequestSize,WritePriority);
            break;
            }
         default:
            printf("WritePolicy not defined for [WriteHit] procedure.");
            exit(1);
         }
      }

   }
```

```
/*****************************************************************************
**                                                          Page  4- 8   **
**                                 Cache.c                                 **
**                                                                         **
**                                WriteMiss                                **
**                                                                         **
**  Description:                                                           **
**                                                                         **
**       WriteMiss is called to simulate a cache miss during a write       **
**  request.  WriteMiss will first simulate the time needed to perform     **
**  all block management requests.  The time is called WriteCacheMissTime. **
**  This is only the time required to make the requests, not the time      **
**  required to complete the block management requests.  The time to       **
**  determine that a miss occurred was simulated by CacheModel.  Once the  **
**  WriteCacheMissTime has expired, then WriteMiss will perform all block  **
**  management requests.  The memory requests are dictated by the          **
**  WriteMissPolicy.  The simplest policy is WriteAround.  For a           **
**  WriteAround policy the write data is placed in the WriteBuffer by      **
**  AddToWriteBuffer.  WriteAllocate however, is the toughest simulation   **
**  in SACS.  WriteMiss must first choose a block to put the new data in.  **
**  This is done by SelectBlockVictim.  Then the block data not provided   **
**  by the write has to be read in.  This read request is made by          **
**  AddToReadBuffer. Because the read address is calculated by adding the  **
**  request size to the address.  The new address may be in the next block **
**  so to make the addition modulo the BlockSize may have to be subtracted. **
**  When the read request has been make then the sub blocks that were      **
**  written to in there entirety will have there valid bits set.  If only  **
**  part of a sub block was written to then the CacheValidBit will not be  **
**  set.                                                                   **
**                                                                         **
**       WriteMiss then uses the WritePolicy to dictate how the write data **
**  is to update the memory.  For a WriteBack policy dirty bits are set by **
**  SetDirtyBits.  For a WriteThough the data is added to the WriteBuffer  **
**  by AddToWriteBuffer.                                                   **
**                                                                         **
*****************************************************************************/


void WriteMiss()

   {

   AddressType TempAddress;

   AccessCache(WriteCacheMissTime, CacheWaitingForWriteCacheRequest);

   switch (WriteMissPolicy)
      {

      case WriteAround:
         {
         if (CacheWaitingFor==Nothing ||
             CacheWaitingFor==CacheWaitingForFullWriteBuffer)
            AddToWriteBuffer(RequestAddress, RequestSize, WritePriority);
         break;
         }

      default:
         {
         printf("WriteMissPolicy not defined in [WriteMiss] procedure");
         exit(1);
         }
```

```
/*****************************************************************************
**                                                          Page  4- 9  **
**                                Cache.c                                  **
**                                                                         **
**                               WriteMiss                                 **
**                               continued                                 **
**                                                                         **
*****************************************************************************/


      case WriteAllocate:
         {

         if (CacheWaitingFor==Nothing ||
             CacheWaitingFor==CacheWaitingForFullWriteBuffer)
            SelectBlockVictim();

         if (CacheWaitingFor==Nothing ||
             CacheWaitingFor==CacheWaitingForFullReadBuffer)
            {
            if ((BlockSize-RequestSize)>0)
               if (BlockAddress(RequestAddress+RequestSize)
                 ==BlockAddress(RequestAddress))
                  AddToReadBuffer(RequestAddress+RequestSize,
                                  BlockSize-RequestSize,
                                  0,
                                  RequestBlockNumber,
                                  ReadForWriteAllocatePriority);
               else
                  AddToReadBuffer((RequestAddress+RequestSize)-BlockSize,
                                  BlockSize-RequestSize,
                                  0,
                                  RequestBlockNumber,
                                  ReadForWriteAllocatePriority);
            }

         if ((CacheWaitingFor==Nothing ||
              CacheWaitingFor==CacheWaitingForFullWriteBuffer) &&
            CacheBlockAddress[RequestBlockNumber]==BlockAddress(RequestAddress))
            {

            for (TempAddress =SubBlockAddress(RequestAddress+SubBlockSize-1);
                 TempAddress <SubBlockAddress(RequestAddress+RequestSize);
                 TempAddress+=SubBlockSize)
               CacheValidBit[RequestBlockNumber][SubBlock(TempAddress)]=Yes;

            switch (WritePolicy)
               {
               case WriteBack:
                  SetDirtyBits();
                  break;
               case WriteThrough:
                  AddToWriteBuffer(RequestAddress, RequestSize, WritePriority);
                  break;
               default:
                  printf("WritePolicy not defined for [WriteMiss] procedure");
                  exit(1);
               }
            }
         break;
         }

      }
   }
```

```
/*****************************************************************************
**                                                          Page  4-10  **
**                                Cache.c                                **
**                                                                       **
**                              AccessCache                              **
**                                                                       **
**  Description:                                                         **
**                                                                       **
**       AccessCache is called to simulate a the CPU accessing the cache.  **
**  AccessCache first waits for the cache not to be busy.  The only reason **
**  it could be busy is if the BlockBuffer is in the process of updating   **
**  the cache.  During this time AccessCache will return CacheWaitingFor   **
**  equal to CPUCacheAccess.  Once the cache is not busy then CacheBusy is **
**  set to Yes locking out the BlockBuffer from accessing the cache.  Then **
**  CacheWaitingFor will set equal to WaitingForRequest this is a local    **
**  variable passed by the caller.  It will either be equal to            **
**  ReadCacheAccess, or WriteCacheAccess.  Then CacheBusy is set for the   **
**  time specified by RequestTime.  RequestTime is a local variable.  It   **
**  could equal any of the hit, miss, or access times.  Once RequestTime   **
**  has expired then AccessCache will set CacheBusy equal to No, and       **
**  CacheWaiting For equal to Nothing.                                     **
**                                                                       **
*****************************************************************************/


void AccessCache(RequestTime,WaitingForRequest)

   TimeType            RequestTime;
   CacheWaitingForType WaitingForRequest;

   {

   static TimeType CacheTOA=0;

   if (CacheBusy==Yes && CacheWaitingFor==Nothing  )
      CacheWaitingFor=CacheWaitingForCPUCacheAccess;
   if (CacheBusy==No  && CacheWaitingFor==CacheWaitingForCPUCacheAccess)
      CacheWaitingFor=Nothing;

   if (CacheWaitingFor==Nothing)
      {
      CacheBusy=Yes;
      CacheWaitingFor=WaitingForRequest;
      CacheTOA=Time+RequestTime;
      }

   RecordStall(CacheWaitingFor);

   if (CacheTOA<=Time && CacheWaitingFor==WaitingForRequest)
      {
      CacheBusy=No;
      CacheWaitingFor=Nothing;
      }

   }
```

```
/*****************************************************************************
**                                                                Page  4-11  **
**                                Cache.c                                    **
**                                                                           **
**                            SelectBlockVictim                              **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       SelectBlockVictim chooses the next block to be used, and writes     **
**  the dirty subblocks out to the WriteBuffer.  SelectBlockVictim first     **
**  surveys the cache set that the RequestAddress maps to.  The servey       **
**  includes finding the block that was least recently accessed.  This      **
**  BlockNumber is stored in LRUBlock.  Once the set has been surveyed       **
**  then the ReplacementPolicy dictates how the block is chosen.  For the    **
**  LRU policy Request Block Number is set equal to LRUBlock.  For the       **
**  FIFO policy CacheNextBlock keeps track of the next victim block for      **
**  each set.  CacheNextBlock is initialized to all zeros during the         **
**  beginning of a run.  Therefore it must be checked to see if it is        **
**  between the first, and last blocks for the set.  If it is not then       **
**  CacheNextBlock for SetNumber is reset to FirstBlock.  Once               **
**  SelectBlockVictim knows it has a valid Cache Next Block then             **
**  RequestBlock is set equal to it.  Then CacheNextBlock for the            **
**  SetNumber is incremented.  For RAND policy the block number is chosen    **
**  randomly from all the blocks in the set                                  **
**                                                                           **
**       SelectBlockVictim writes all dirty sub blocks to the WriteBuffer    **
**  using WriteDirtySubBlocks.  WriteDirtySubBlocks takes care of clearing   **
**  the dirty and valid bits in the block.  Once SelectBlockVictim is        **
**  called and it gets to the bottom of the function with CacheWaitingFor    **
**  equal to Nothing then the CacheBlockAddress for the RequestBlockNumber   **
**  is set equal to the block address of RequestAddress.                     **
**                                                                           **
*****************************************************************************/


void SelectBlockVictim()

   {

   SizeType   SetNumber     = Set(RequestAddress);
   SizeType   FirstBlock    = SetNumber*Associativity;
   SizeType   LastBlock     = FirstBlock+Associativity-1;
   SizeType   BlockIndex;
   SizeType   SubBlockIndex;

   TimeType   LRUTime       = Time+1;
   SizeType   LRUBlock;
```

```
/*****************************************************************************
**                                                                          **
**                                Cache.c                                   **
**                                                                          **
**                            SelectBlockVictim                             **
**                               Continued                                  **
**                                                                          **
*****************************************************************************/

   RequestBlockNumber=FirstBlock;

   for (BlockIndex=FirstBlock; BlockIndex<=LastBlock; BlockIndex++)
      {
      if (CacheBlockAddress[BlockIndex]==BlockAddress(RequestAddress))
         RequestBlockNumber=BlockIndex;
      if (LRUTime>LastCacheBlockAccessTime[BlockIndex])
         {
         LRUTime=LastCacheBlockAccessTime[BlockIndex];
         LRUBlock=BlockIndex;
         }
      }

   if (CacheBlockAddress[RequestBlockNumber]!=BlockAddress(RequestAddress)

      {

      switch (BlockReplacementPolicy)

         {

         case LRU:
            RequestBlockNumber=LRUBlock;
            LastCacheBlockAccessTime[RequestBlockNumber]=Time;
            break;

         case FIFO:
            if (CacheNextBlock[SetNumber]<FirstBlock ||
                 CacheNextBlock[SetNumber]>LastBlock    )
               CacheNextBlock[SetNumber]=FirstBlock;
            RequestBlockNumber=CacheNextBlock[SetNumber];
            if (RequestBlockNumber<LastBlock)
               CacheNextBlock[SetNumber]++;
            else
               CacheNextBlock[SetNumber]=FirstBlock;
            break;

         case RAND:
            RequestBlockNumber=(rand()%Associativity)+FirstBlock;
            break;

         }

      WriteDirtySubBlocks();

      }

   if (CacheWaitingFor==Nothing)
      CacheBlockAddress[RequestBlockNumber]=BlockAddress(RequestAddress);

   }
```

```
/*****************************************************************************
**                                                              Page  4-13  **
**                                 Cache.c                                  **
**                                                                          **
**                               SetDirtyBits                               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       SetDirtyBits sets the dirty bits for all sub blocks that contains  **
**  data that was modified by a write request.                              **
**                                                                          **
*****************************************************************************/


void SetDirtyBits()

   {

   SizeType SubBlockIndex;

   for (SubBlockIndex=SubBlock(RequestAddress);
        SubBlockIndex<=SubBlock(RequestAddress+RequestSize-1);
        SubBlockIndex++)
      CacheDirtyBit[RequestBlockNumber][SubBlockIndex]=Yes;

   }
```

```
/*****************************************************************************
**                                                              Page  4-14  **
**                                 Cache.c                                  **
**                                                                          **
**                            WriteDirtySubBlocks                           **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       WriteDirtySubBlocks is called to simulate writing all the dirty    **
**  sub blocks in RequestBlock.  WriteDirtySubBlocks not only clears all    **
**  the dirty bits.  It also clears all the valid bits.                     **
**  WriteDirtySubBlocks prepares a block to receive new data, and is called **
**  after a block has been selected as a victim.  WriteDirtySubBlocks will  **
**  search the block for consecutive dirty blocks and splice them together  **
**  into one write request.  The write request is then added to the         **
**  WriteBuffer.  All of the sub blocks that make up the request will have  **
**  their dirty and valid bits cleared.  This process of searching and      **
**  writing is repeated until all the bits are not dirty.  Then all the     **
**  valid bits are cleared.                                                 **
**                                                                          **
*****************************************************************************/

void WriteDirtySubBlocks()

   {

   SizeType     i;
   SizeType     SubBlockIndex         = 0;

   AddressType  MemoryRequestAddress  = CacheBlockAddress[RequestBlockNumber];
   SizeType     MemoryRequestSize     = 0;
   PriorityType MemoryRequestPriority = WriteDirtyBlockPriority;
```

```
/*****************************************************************************
**                                                                          **
**                                Cache.c                                   **
**                                                                          **
**                           WriteDirtySubBlocks                            **
**                               continued                                  **
**                                                                          **
*****************************************************************************/


   do
      {

      MemoryRequestSize=0;

      while ((CacheDirtyBit[RequestBlockNumber][SubBlockIndex]==No  ||
              CacheValidBit[RequestBlockNumber][SubBlockIndex]==No) &&
             SubBlockIndex<NumberOfSubBlocks)
         SubBlockIndex++;

      MemoryRequestAddress=CacheBlockAddress[RequestBlockNumber]
                          +SubBlockIndex*SubBlockSize;

      while (CacheDirtyBit[RequestBlockNumber][SubBlockIndex]==Yes &&
             SubBlockIndex<NumberOfSubBlocks)
         {
         MemoryRequestSize+=WordSize;
         SubBlockIndex++;
         }

      if (MemoryRequestSize)
         {
         AddToWriteBuffer(MemoryRequestAddress,
                          MemoryRequestSize,
                          MemoryRequestPriority);
         if (CacheWaitingFor==Nothing)
            for (i=0; i<=SubBlockIndex && i<NumberOfSubBlocks; i++)
               CacheDirtyBit[RequestBlockNumber][i]=No;
         }

      }
   while (SubBlockIndex<NumberOfSubBlocks && CacheWaitingFor==Nothing);

   if (CacheWaitingFor==Nothing)
      for (i=0; i<NumberOfSubBlocks; i++)
         CacheValidBit[RequestBlockNumber][i]=No;

   }
```

Cache.c

AddToReadBuffer

Description:

AddToReadBuffer takes the elements of a request, and adds the request to the ReadBuffer. It will perform all of the searches, and updates necessary to support the appropriate scoreboarding protocals.

AddToReadBuffer will begin by searching the cache, and BlockBuffer for each byte in the request starting at the beginning of the request. Every time a byte is found in one or the other then the Address is incremented, while Size and RequiredSize are decremented. This simulates removing the available data from the front of the request. Then AddToReadBuffer will search the cache, and BlockBuffer for the data at the end of the request. Every time a byte is found then the Size of the request is decremented by one. If the byte was a required byt then the RequiredSize is decremented also. This simulates removing any data available from the end of the request. AddToReadBuffer is either left with a request that has a Size equal to zero or the end points are both needed from memory. If the RequiredSize is zero then the request is a buffer hit, otherwize the request is a buffer miss. If the request is already a cache hit then the buffer hit is for some block management request. These kinds of buffer hits are not recorded because it would confuse the ResultsDisplay, by making it possible to get a hit rate greater tha 100%. If the Size is not zero and Remove ReadDuplicates is eaual to No then the request is added to the end of the ReadBuffer using Append. Append is a buffer utility that adds the request to the end of the buffer. The request must be added to the end of the buffer in ouder not to interfere with MemoryModel which maybe in the middle of a memory read. If RemoveReadDuplicates is equal to Yes then the first byte in the request will be spliced into the Read Buffer.

Splice is another buffer utility. Splice will first search the ReadBuffer for the byte if it cant't find a request in the buffer that contains the byte then it will search for a read request that is getting data from the same block. If one is found then the request is modified to include the new read byte request. If no suitable request can be found then Splice will add a one byt request to the Read BUffer. The Address is then incremented while the Size, and Required Size are decremented. Then the cache, and BlockBuffer are searched for the next byte. If it is not found then the next byte is spliced into the ReadBuffer. This process is repeated until all of the bytes of the request have either been spliced into the ReadBuffer or found.

The BufferHit is normally defined as when the data is available but in the cache. However in order to support the testing of SACS, the definition of a buffer hit is redefined to mean that a request was found to have accrued recently, and that given time to complete all block management the requested data would have been in the cache. This allows TestSACS to predict the hits of a test run without taking into account the time in takes to preform the block management.

Every time a request is spliced into the read or write buffers then the TimeToExecute, and CompletionTimeExtamate must be recalculated. The new time estimates are performed by CalculateTimeEstimates.

```
/*****************************************************************************
**                                                          Page  4-17   **
**                                Cache.c                                  **
**                                                                         **
**                            AddToReadBuffer                              **
**                               continued                                 **
**                                                                         **
*****************************************************************************/


void AddToReadBuffer(Address,Size,RequiredSize,Block,Priority)

   AddressType  Address;
   SizeType     Size;
   SizeType     RequiredSize;
   SizeType     Block;
   PriorityType Priority;

   {

   MemoryRequestType ReadMemoryRequest;
   YesNoType         FoundByte;
   AddressType       ByteAddress;
   AddressType       CurrentBlockAddress    = BlockAddress(Address);
   BufferSizeType    OldReadBufferNext      = ReadBuffer.Next;

   ReadMemoryRequest.Address                = Address;
   ReadMemoryRequest.Size                   = Size;
   ReadMemoryRequest.RequiredSize           = RequiredSize;
   ReadMemoryRequest.Block                  = Block;
   ReadMemoryRequest.Priority               = Priority;
   ReadMemoryRequest.AccessInProgress       = No;
   ReadMemoryRequest.TimeToExecute          = 0;
   ReadMemoryRequest.CompletionTimeEstimate = 0;
```

```
/*****************************************************************************
**                                                                          **
**                                Cache.c                                   **
**                                                                          **
**                            AddToReadBuffer                               **
**                               continued                                  **
**                                                                          **
*****************************************************************************/

   if (CacheWaitingFor==CacheWaitingForFullReadBuffer) CacheWaitingFor=Nothing;

   FoundByte=Yes;
   while (FoundByte==Yes && Size>0)
      {
      FoundByte=No;
      if (SearchCache(Address)==Yes)
         FoundByte=Yes;
      else if (SearchBlockBuffer==Yes && Search(&BlockBuffer, Address))
         FoundByte=Yes;
      if (FoundByte==Yes)
         {
         Address++;
         if (BlockAddress(Address)!=CurrentBlockAddress) Address-=BlockSize;
         if (Size>0)    Size--;
         if (RequiredSize>0) RequiredSize--;
         }
      }

   ByteAddress=Address+Size-1;
   if (BlockAddress(ByteAddress)!=CurrentBlockAddress) ByteAddress-=BlockSize;
   FoundByte=Yes;
   while (FoundByte==Yes && Size>0)
      {
      FoundByte=No;
      if (SearchCache(ByteAddress)==Yes)
         FoundByte=Yes;
      else if (SearchBlockBuffer==Yes && Search(&BlockBuffer, ByteAddress))
         FoundByte=Yes;
      if (FoundByte==Yes)
         {
         ByteAddress--;
         if (BlockAddress(ByteAddress)!=CurrentBlockAddress)
            ByteAddress+=BlockSize;
         if (Size>0)       Size--;
         if (RequiredSize>Size) RequiredSize=Size;
         }
      }

   if (Request==Read && Test==No)
      {
      if (RequiredSize==0 && CacheHit==No)
         BufferHit=Yes;
      else
         BufferHit=No;
      }

   if (RequiredSize==0 && Request==Read) CacheWaitingFor=Nothing;

   ReadMemoryRequest.Address      = Address;
   ReadMemoryRequest.Size         = Size;
   ReadMemoryRequest.RequiredSize = RequiredSize;

   if (RemoveReadDuplicates==No && Size>0)
      Append(&ReadBuffer,&ReadMemoryRequest);
```

```
/*****************************************************************************
**                                                                          **
**                                Cache.c                                   **
**                                                                          **
**                            AddToReadBuffer                               **
**                               continued                                  **
**                                                                          **
*****************************************************************************/


   while (Size>0 && RemoveReadDuplicates==Yes)
      {
      FoundByte=No;
      if (SearchCache(Address)==Yes)
         FoundByte=Yes;
      else if (SearchBlockBuffer==Yes && Search(&BlockBuffer, Address))
         FoundByte=Yes;

      if (FoundByte==No)
         Splice(&ReadBuffer,Address,RequiredSize,Block,Priority);

      Address++;
      if (BlockAddress(Address)!=CurrentBlockAddress) Address-=BlockSize;
      if (Size>0)    Size--;
      if (RequiredSize>0) RequiredSize--;

      }

   if (Request==Read && Test==Yes)
      {
      if (ReadBuffer.Next==OldReadBufferNext && CacheHit==No)
         BufferHit=Yes;
      else
         BufferHit=No;
      }

   CalculateTimeEstimates();

   }
```

```
/*****************************************************************************
**                                                          Page  4-20   **
**                                Cache.c                                 **
**                                                                        **
**                              SearchCache                               **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       SearchCache is called by AddToReadBuffer to find any parts of    **
**  the request that may be already located in the cache.  This must be   **
**  done because if a read request follows a write request using a write  **
**  allocate policy then part of the read may be in the cache while the   **
**  rest may still need to be read from memory.  Search Cache checks all  **
**  CacheBlockAddresses in the cache set.  If any of the cache block      **
**  addresses equals the block address of the byte, then Search Cache     **
**  checks the CacheValidBit for the sub block that the byte is located in. **
**  If the sub block is valid then SearchCache returns Yes.               **
**                                                                        **
*****************************************************************************/

YesNoType SearchCache(Address)

   AddressType Address;

   {

   SizeType  FirstBlock = Set(Address)*Associativity;
   SizeType  LastBlock  = FirstBlock+Associativity-1;
   SizeType  BlockIndex;
   YesNoType FoundByte;


   for (BlockIndex=FirstBlock; BlockIndex<=LastBlock; BlockIndex++)
      if (CacheBlockAddress[BlockIndex]==BlockAddress(Address))
         if (CacheValidBit[BlockIndex][SubBlock(Address)]) FoundByte=Yes;

   return(FoundByte);

   }
```

```
/*****************************************************************************
**                                                              Page  4-21  **
**                                 Cache.c                                  **
**                                                                          **
**                             AddToWriteBuffer                             **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       AddToWriteBuffer adds one record to write buffer.  It also updates **
**  the ReadBuffer it the UpdateReadBuffer arguments is asserted.  The      **
**  process of updating the ReadBuffer is simply changein the requests so   **
**  that data make available by the write request is not requested form     **
**  memory.  UpdateReadBuffer should not be used unless the word and sub    **
**  block sizes are equal.  This is because a write request may reduce a    **
**  read request to where the read request will not be large enough to      **
**  validate a sub block.  The write request may alsobe unable to set any   **
**  valid bits because of sub block alignment.  The result is that a sub    **
**  block was supposed to be read in is not.                                **
**                                                                          **
*****************************************************************************/


void AddToWriteBuffer(Address,Size,Priority)

   AddressType  Address;
   SizeType     Size;
   PriorityType Priority;

   {

   MemoryRequestType WriteMemoryRequest;
   YesNoType         FoundByte;
   AddressType       ByteAddress;
   AddressType       CurrentBlockAddress = BlockAddress(Address);
   SizeType          NoBytes;
   BufferSizeType    OldWriteBufferNext  = WriteBuffer.Next;

   WriteMemoryRequest.Address                = Address;
   WriteMemoryRequest.Size                   = Size;
   WriteMemoryRequest.RequiredSize           = 0;
   WriteMemoryRequest.Block                  = 0;
   WriteMemoryRequest.Priority               = Priority;
   WriteMemoryRequest.AccessInProgress       = No;
   WriteMemoryRequest.TimeToExecute          = 0;
   WriteMemoryRequest.CompletionTimeEstimate = 0;
```

```
/*****************************************************************************
**                                                                Page  4-22  **
**                                Cache.c                                    **
**                                                                           **
**                            AddToWriteBuffer                               **
**                               continued                                   **
**                                                                           **
*****************************************************************************/


   if (CacheWaitingFor==CacheWaitingForFullWriteBuffer)
      CacheWaitingFor=Nothing;

   FoundByte=Yes;
   while(FoundByte==Yes && UpdateReadBuffer==Yes)
      {
      FoundByte=No;
      ByteAddress=Address;
      for (NoBytes=0; NoBytes<Size; NoBytes++)
         {
         if (UpdatingReadBuffer(ByteAddress)==Yes) FoundByte=Yes;
         ByteAddress++;
         if (BlockAddress(ByteAddress)!=CurrentBlockAddress)
            ByteAddress-=BlockSize;
         }
      }

   if (RemoveWriteDuplicates==No && Size>0)
      Append(&WriteBuffer, &WriteMemoryRequest);

   while (RemoveWriteDuplicates==Yes && Size>0)
      {
      Splice(&WriteBuffer,Address,0,0,Priority);
      Address++;
      if (BlockAddress(Address)!=CurrentBlockAddress) Address-=BlockSize;
      if (Size>0) Size--;
      }

   if (Request==Write)
      {
      BufferHit=No;
      if (WriteBuffer.Next==OldWriteBufferNext && CacheHit==No) BufferHit=Yes;
      if (WriteBuffer.Next==OldWriteBufferNext &&
          CacheWaitingFor !=CacheWaitingForFullWriteBuffer)
         CacheWaitingFor=Nothing;
      }

   CalculateTimeEstimates();

   }
```

```
/*****************************************************************************
**                                                          Page  5- 0      **
**                              Memory.c                                    **
**                                                                          **
**                          Part Of SACS 1.0                                **
**                    (StillAnother Cache Simulator)                        **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                              Page  5- 1  **
**                                 Memory.c                                 **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Memory.c contains all functions that relate to the simulation of   **
**  main memory. Memory Model makes all the necessary calls to simulate     **
**  main memory.  MemoryModel decides which calls to make, based on         **
**  MemoryWaitingFor.  This function is called every time Time is           **
**  incremented.  If there are no read or write requests waiting to be      **
**  completed, the function does nothing.  Memory Model contains a loop     **
**  that forces the procedure to continue modeling until TOA and TOD are    **
**  not equal to Time.  This insures that if there are any events that      **
**  occur in zero clock cycles then the next event is allowed to start.     **
**                                                                          **
**       Memory Model calls SelectMemoryRequest to choose a request from    **
**  either the read or the write buffers.  Memory Model calls Start Reads,  **
**  and Start Writes, to simulate accessing memory and receiving the first  **
**  word of a memory request.  ContinueMemoryReads, and                     **
**  ContinueMemoryWrites are then called to simulate the memory transfer    **
**  of the following words of data.                                         **
**                                                                          **
**       The simulation of main memory includes:                            **
**                                                                          **
**            Choosing memory request from read, write buffers.             **
**            Simulated memory access times.                                **
**            Simulated memory transfer times.                              **
**            Cache Update after memory read.                               **
**                                                                          **
**                             Table of Contents                            **
**                                                                          **
**          Cover Page ............................... Page  5- 1          **
**          List of Memory.c Function Declarations .... Page  5- 2          **
**          MemoryModel() ............................ Page  5- 3          **
**             SelectMemoryRequest.................... Page  5- 4          **
**             StartMemoryReads() .................... Page  5- 5          **
**             ContinueMemoryReads() ................. Page  5- 6          **
**             StartMemoryWrites() ................... Page  5- 8          **
**             ContinueMemoryWrites() ................ Page  5- 9          **
**          UpdateCache() ............................ Page  5-11          **
**          AddAWordToMemoryRequest() ................ Page  5-13          **
**          RemoveAWordFromMemoryRequest() ........... Page  5-14          **
**                                                                          **
*****************************************************************************/

#include "Global.h"
```

```
/*****************************************************************************
**                                                              Page  5- 2  **
**                                 Memory.c                                 **
**                                                                          **
**                List of Memory.c Function Declarations                    **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of Memory.c                                                             **
**                                                                          **
*****************************************************************************/


void MemoryModel();                                          /* Page  5- 3 */
void    SelectMemoryRequest();                               /* Page  5- 4 */
void    StartMemoryReads();                                  /* Page  5- 5 */
void    ContinueMemoryReads();                               /* Page  5- 6 */
void    StartMemoryWrites();                                 /* Page  5- 8 */
void    ContinueMemoryWrites();                              /* Page  5- 9 */
void UpdateCache();                                          /* Page  5-11 */
void AddAWordToMemoryRequest();                              /* Page  5-13 */
void RemoveAWordFromMemoryRequest();                         /* Page  5-14 */
```

```
/*****************************************************************************
**                                                          Page  5- 3  **
**                                Memory.c                                **
**                                                                        **
**                               MemoryModel                              **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       Memory.c contains all functions that relate to the simulation of **
**  main memory. Memory Model makes all the necessary calls to simulate   **
**  main memory.  MemoryModel decides which calls to make, based on       **
**  MemoryWaitingFor.  This function is called every time Time is         **
**  incremented.  If there are no read or write requests waiting to be    **
**  completed, the function does nothing.  Memory Model contains a loop   **
**  that forces the procedure to continue modeling until TOA and TOD are  **
**  not equal to Time.  This insures that if there are any events that    **
**  occur in zero clock cycles then the next event is allowed to start.   **
**                                                                        **
**       Memory Model calls SelectMemoryRequest to choose a request from  **
**  either the read or the write buffers.  Memory Model calls Start Reads, **
**  and Start Writes, to simulate accessing memory and receiving the first **
**  word of a memory request.  ContinueMemoryReads, and                   **
**  ContinueMemoryWrites are then called to simulate the memory transfer  **
**  of the following words of data.                                       **
**                                                                        **
**       The simulation of main memory includes:                          **
**                                                                        **
**            Choosing memory request from read, write buffers.           **
**            Simulated memory access times.                              **
**            Simulated memory transfer times.                            **
**            Cache Update after memory read.                             **
**                                                                        **
*****************************************************************************/

void MemoryModel()

   {

   MemoryWaitingForType LastMemoryWaitingFor;

   do
      {
      LastMemoryWaitingFor=MemoryWaitingFor;

      if (MemoryWaitingFor==Nothing) SelectMemoryRequest(&MemoryWaitingFor);

      else if (MemoryWaitingFor==MemoryWaitingForMemoryReadRequest ||
               MemoryWaitingFor==MemoryWaitingForCacheUpdate)
              StartMemoryReads();

      else if (MemoryWaitingFor==MemoryWaitingForMemoryReadAccess  ||
               MemoryWaitingFor==MemoryWaitingForMemoryReadTransfer)
              ContinueMemoryReads();

      else if (MemoryWaitingFor==MemoryWaitingForMemoryWriteRequest)
              StartMemoryWrites();

      else if (MemoryWaitingFor==MemoryWaitingForMemoryWriteAccess ||
               MemoryWaitingFor==MemoryWaitingForMemoryWriteTransfer)
              ContinueMemoryWrites();
      }
   while (MemoryWaitingFor!=LastMemoryWaitingFor || TOA==Time || TOD==Time);

   }
```

```
/*****************************************************************************
**                                                          Page  5- 4   **
**                                Memory.c                                **
**                                                                        **
**                           SelectMemoryRequest                          **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       SelectMemoryRequest is called when memory is waiting for nothing. **
**  SelectMemoryRequest chooses a request from either the read or write   **
**  buffers, based on priority.  The request is not returned however, the **
**  request is left at the top of the buffer with its Priority and       **
**  AccessInprogress set equal to Yes.  If a request is found then       **
**  MemoryWaitingFor is set to MemoryReadRequest, or MemoryWriteRequest   **
**  depending on whether the request was found in the read or write      **
**  buffers.                                                              **
**                                                                        **
*****************************************************************************/


void SelectMemoryRequest(MemoryWaitingFor)

   MemoryWaitingForType *MemoryWaitingFor;

   {

   MemoryRequestType ReadMemoryRequest;
   MemoryRequestType WriteMemoryRequest;

   if (!(ReadBuffer.Empty))
      ReadMemoryRequest=View(&ReadBuffer);
   else
      ReadMemoryRequest.Priority=NoPriority;

   if (!(WriteBuffer.Empty))
      WriteMemoryRequest=View(&WriteBuffer);
   else
      WriteMemoryRequest.Priority=NoPriority;

   if (ReadMemoryRequest.Priority<=WriteMemoryRequest.Priority &&
       ReadMemoryRequest.Priority!=NoPriority)
      {
      *MemoryWaitingFor=MemoryWaitingForMemoryReadRequest;
      ReadMemoryRequest.AccessInProgress=Yes;
      ReadMemoryRequest.Priority=0;
      ChangeTopMemoryRequest(&ReadBuffer,&ReadMemoryRequest);
      }
   else if (WriteMemoryRequest.Priority!=NoPriority)
      {
      *MemoryWaitingFor=MemoryWaitingForMemoryWriteRequest;
      WriteMemoryRequest.AccessInProgress=Yes;
      WriteMemoryRequest.Priority=0;
      ChangeTopMemoryRequest(&WriteBuffer,&WriteMemoryRequest);
      }

   }
```

```
/*****************************************************************************
**                                                          Page  5- 5  **
**                                Memory.c                                **
**                                                                        **
**                            StartMemoryReads                            **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       StartMemoryReads begins a read request, simulating the first word **
**  read from memory.  The time to complete this read is called          **
**  MemoryAccessTime.  The BlockBuffer is initialized in preparation to   **
**  receive the new data words.  If BlockWaitingFor is not equal to       **
**  Nothing the StartMemoryReads will have to wait until it is before     **
**  allowing the new memory read request to start.  If StartMemoryReads  **
**  does have to wait for the cache then MemoryWaitingFor it set equal to **
**  CacheUpdate, otherwise MemoryWaitingFor is set to MemoryReadAccess.   **
**  The new block record is equal to the ReadBuffer with its sizes set to **
**  zero.  This gives the Block Memory Request the same block number and  **
**  the ReadMemoryRequest.  The Address is aligned to WordSize.  The      **
**  Address must be aligned because the words read in will be aligned     **
**  to WordSize.  The new BlockMemoryRequest is simply pushed onto the    **
**  Block Buffer. The BlockWaitingFor is set equal to MemoryBlockTransfer. **
**  To indicate that data is being transferred from memory to the         **
**  BlockBuffer.                                                          **
**                                                                        **
*****************************************************************************/


void StartMemoryReads()

   {

   MemoryRequestType ReadMemoryRequest;
   MemoryRequestType BlockMemoryRequest;

   if (BlockWaitingFor==Nothing)
      {

      ReadMemoryRequest=View(&ReadBuffer);

      TOA=Time+MemoryAccessTime;
      MemoryWaitingFor=MemoryWaitingForMemoryReadAccess;

      BlockMemoryRequest=ReadMemoryRequest;

      BlockMemoryRequest.Address=WordAddress(ReadMemoryRequest.Address);
      BlockMemoryRequest.Size=0;
      BlockMemoryRequest.RequiredSize=0;
      BlockMemoryRequest.Priority=0;
      BlockMemoryRequest.AccessInProgress=No;

      Push(&BlockBuffer,&BlockMemoryRequest);

      BlockWaitingFor=MemoryBlockTransfer;

      }
   else
      {

      MemoryWaitingFor=MemoryWaitingForCacheUpdate;

      }

   }
```

```
/*****************************************************************************
**                                                          Page  5- 6  **
**                                Memory.c                                **
**                                                                        **
**                          ContinueMemoryReads                           **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       ContinueMemoryReads continues the memory read request started by      **
**  StartMemoryReads.  It simulates every read from memory other than the  **
**  first word which was simulated by StartMemoryReads.  The time to       **
**  complete each word transfer is equal to MemroyTransferTime.  The block, **
**  and read buffers are altered every time a word is read from memory.    **
**  Once a request is complete, it is removed from the Read Buffer, and    **
**  Memory WaitingFor is reset to Nothing.  Block Waiting For is set to    **
**  BlockCacheAccess in preparation to transfer the new data to the cache.  **
**  If the CompletionTimeEstimate for the memory read request is not equal **
**  to Time then a time predition error is rased.                         **
**                                                                        **
*****************************************************************************/


void ContinueMemoryReads()

   {

   MemoryRequestType BlockMemoryRequest;
   MemoryRequestType ReadMemoryRequest;

   if (TOA<=Time)

      {

      BlockMemoryRequest=View(&BlockBuffer);
      AddAWordToMemoryRequest(&BlockMemoryRequest);
      ChangeTopMemoryRequest(&BlockBuffer,&BlockMemoryRequest);

      ReadMemoryRequest=View(&ReadBuffer);

      RemoveAWordFromMemoryRequest(&ReadMemoryRequest);

      if (ReadMemoryRequest.Size>0)
         {
         ChangeTopMemoryRequest(&ReadBuffer,&ReadMemoryRequest);
         TOA=Time+MemoryTransferTime;
         MemoryWaitingFor=MemoryWaitingForMemoryReadTransfer;
         }
      else
         {
         Pop(&ReadBuffer);
         TOA=0;
         if (Time!=ReadMemoryRequest.CompletionTimeEstimate)
            PrintTimePredictionError(ReadMemoryRequest.CompletionTimeEstimate,
                                     Time,
                                     "Read",
                                     "ContinueMemoryReads");
         MemoryWaitingFor=Nothing;
         BlockWaitingFor=BlockCacheAccess;
         BlockTOA=Time+BufferCacheAccessTime;
         }

      TotalNumberOfWordsReadFromMemory++;

      }
```

```
/*****************************************************************************
**                                                                          **
**                                Memory.c                                  **
**                                                                          **
**                          ContinueMemoryReads                             **
**                               continued                                  **
**                                                                          **
*****************************************************************************/

   else

      {

      ReadMemoryRequest=View(&ReadBuffer);

      if (ReadMemoryRequest.Size==0)
         {
         Pop(&ReadBuffer);
         TOA=0;
         if (Time!=ReadMemoryRequest.CompletionTimeEstimate)
            PrintTimePredictionError(ReadMemoryRequest.CompletionTimeEstimate,
                                     Time,
                                     "Read",
                                     "ContinueMemoryReads");
         MemoryWaitingFor=Nothing;
         BlockWaitingFor=BlockCacheAccess;
         BlockTOA=Time+BufferCacheAccessTime;
         }

      }

   }
```

```
/*****************************************************************************
**                                                              Page  5- 8  **
**                                 Memory.c                                 **
**                                                                          **
**                             StartMemoryWrites                            **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       StartMemoryWrites begins a memory write request, simulating the    **
**  first word written to memory.  The time to complete this one word write **
**  is called MemoryAccessTime.  MemoryWaitingFor is set to                 **
**  MemoryWriteAccess.                                                      **
**                                                                          **
*****************************************************************************/


void StartMemoryWrites()

   {

   if (MemoryWaitingFor==MemoryWaitingForMemoryWriteRequest)
      {
      TOD=Time+MemoryAccessTime;
      MemoryWaitingFor=MemoryWaitingForMemoryWriteAccess;
      }

   }
```

```
/*****************************************************************************
**                                                          Page  5- 9   **
**                                Memory.c                                **
**                                                                        **
**                          ContinueMemoryWrites                          **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       ContinueMemoryWrites continues the memory write request started  **
**  by StartMemoryWrites.  Like ContinueMemoryReads, it simulates every   **
**  write to mwmory other than the first word which was simulated by     **
**  StartMemroyWrites.  The time to complete each word transfer is equal  **
**  to MemoryTransferTime.  The Write Buffer is altered every time a word **
**  is written to memory.  Once the memory write request is complete, it  **
**  is removed form the WriteBuffer, and MemoryWaitingFor is reset to     **
**  Nothing.  If the CompletionTimeEstimate for the memory read request is **
**  not equal to Time when the request is completed then a time predition **
**  error is rased.                                                       **
**                                                                        **
*****************************************************************************/


void ContinueMemoryWrites()

   {
   MemoryRequestType WriteMemoryRequest;

   if (TOD<=Time)
      {
      WriteMemoryRequest=View(&WriteBuffer);
      RemoveAWordFromMemoryRequest(&WriteMemoryRequest);
      if (WriteMemoryRequest.Size>0)
         {
         ChangeTopMemoryRequest(&WriteBuffer,&WriteMemoryRequest);
         TOD=Time+MemoryTransferTime;
         MemoryWaitingFor=MemoryWaitingForMemoryWriteTransfer;
         }
      else
         {
         Pop(&WriteBuffer);
         TOD=0;
         if (Time!=WriteMemoryRequest.CompletionTimeEstimate)
            PrintTimePredictionError(WriteMemoryRequest.CompletionTimeEstimate,
                                     Time,
                                     "Write",
                                     "ContinueMemoryWrites");
         MemoryWaitingFor=Nothing;
         }
      TotalNumberOfWordsWrittenToMemory++;
      }
```

```
/*****************************************************************************
**                                                              Page  5-10  **
**                                 Memory.c                                 **
**                                                                          **
**                           ContinueMemoryWrites                           **
**                                continued                                 **
**                                                                          **
*****************************************************************************/

   else
      {
      WriteMemoryRequest=View(&WriteBuffer);
      if (WriteMemoryRequest.Size==0)
         {
         Pop(&WriteBuffer);
         TOD=0;
         if (Time!=WriteMemoryRequest.CompletionTimeEstimate)
            PrintTimePredictionError(WriteMemoryRequest.CompletionTimeEstimate,
                                     Time,
                                     "Write",
                                     "ContinueMemoryWrites");
         MemoryWaitingFor=Nothing;
         }
      }
   }
```

```
/*****************************************************************************
**                                                          Page  5-11   **
**                              Memory.c                                  **
**                                                                        **
**                             UpdateCache                                **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       UpdateCache simulates entering data from the BlockBuffer into the **
**  cache.  UpdateCache first checks wheter of not the cache is busy.  If **
**  it is not then CacheBusy is asserted, and BlockWaitingFor is set equal **
**  to BlockCacheTransfer.  The BlockTOA is calculated, to enable         **
**  CalculateTimeEstimates to predict the completion times for additional **
**  memory read request in the buffer.  If the cache is busy then the    **
**  previous memory request time completions may be wrong.  That is because **
**  the last estimates conunted on the old BlockTOA.  This means that all **
**  the time estimates must be recalculated.                              **
**                                                                        **
**       Once the BufferCacheAccessTime has expired then BlockWaitingFor  **
**  is set equal to Nothing, and the CacheBusy is deserted.  The read data **
**  must then be removed from the BlockBuffer.  The appropriate sub blocks **
**  in the cache will then have there dirty bits cleared, and valid bits  **
**  set.                                                                  **
**                                                                        **
*****************************************************************************/


void UpdateCache()

   {

   MemoryRequestType BlockMemoryRequest;
   AddressType       TempAddress;
```

```
/*****************************************************************************
**                                                                          **
**                                Memory.c                                  **
**                                                                          **
**                               UpdateCache                                **
**                                continued                                 **
**                                                                          **
*****************************************************************************/

   if (BlockWaitingFor==BlockCacheAccess && CacheBusy==Yes)
      {
      BlockTOA=Time+BufferCacheAccessTime+1;
      CalculateTimeEstimates();
      }

   if (BlockWaitingFor==BlockCacheAccess && CacheBusy==No)
      {
      CacheBusy=Yes;
      BlockTOA=Time+BufferCacheAccessTime;
      BlockWaitingFor=BlockCacheTransfer;
      }

   if (BlockWaitingFor==BlockCacheTransfer && BlockTOA<=Time)
      {
      CacheBusy=No;
      BlockWaitingFor=Nothing;
      BlockTOA=0;
      BlockMemoryRequest=Pop(&BlockBuffer);
      if (CacheBlockAddress[BlockMemoryRequest.Block]==
          BlockAddress(BlockMemoryRequest.Address))
         {
         for (TempAddress=BlockMemoryRequest.Address;
              TempAddress<=BlockMemoryRequest.Address
                          +BlockMemoryRequest.Size-1;
              TempAddress+=SubBlockSize)
            {
            CacheDirtyBit[BlockMemoryRequest.Block][SubBlock(TempAddress)]=No;
            CacheValidBit[BlockMemoryRequest.Block][SubBlock(TempAddress)]=Yes;
            }
         }
      }
   else
      {
      BlockMemoryRequest=View(&BlockBuffer);
      BlockMemoryRequest.TimeToExecute=BufferCacheAccessTime;
      BlockMemoryRequest.CompletionTimeEstimate=BlockTOA;
      ChangeTopMemoryRequest(&BlockBuffer,&BlockMemoryRequest);
      }

   }
```

```
/*****************************************************************************
**                                                          Page  5-13   **
**                              Memory.c                                  **
**                                                                        **
**                       AddAWordToMemoryRequest                          **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       AddAWordToMemoryRequest adds a word to a MemoryRequest as if it  **
**  had been read in from memory.  The address is first aligned to       **
**  WordSize.  This simulates the data being added to the request.       **
**                                                                        **
*****************************************************************************/


void AddAWordToMemoryRequest(MemoryRequest)

   MemoryRequestType *MemoryRequest;

   {

   MemoryRequest->Address=WordAddress(MemoryRequest->Address);

   MemoryRequest->Size+=WordSize;

   }
```

```
/*****************************************************************************
**                                                          Page  5-14   **
**                                Memory.c                                **
**                                                                        **
**                       RemoveAWordFromMemoryRequest                     **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       RemoveAWordFromMemoryRequest removes a word from a Memory Request, **
**  as if it had been written to memory.  A copy of the Address is first   **
**  stored in OldAddress.  Then the Address is word aligned and incremented **
**  by WordSize.  The Required Size, and Size are then decremented by the  **
**  difference of the new Address, and the OldAddress.  Finally if the     **
**  Address is outside the range of the original block then the Address is **
**  decremented by BlockSize to simulate modulo addition.  This simulates  **
**  removing a word from the memory request taking into account word and   **
**  block alignment constraints.                                          **
**                                                                        **
*****************************************************************************/


void RemoveAWordFromMemoryRequest(MemoryRequest)

   MemoryRequestType *MemoryRequest;

   {

   AddressType OldAddress=MemoryRequest->Address;

   MemoryRequest->Address=WordAddress(MemoryRequest->Address)+WordSize;

   if (MemoryRequest->Size>BlockSize-WordSize)
      MemoryRequest->Size=BlockSize-WordSize;
   else if (MemoryRequest->Size>MemoryRequest->Address-OldAddress)
      MemoryRequest->Size-=MemoryRequest->Address-OldAddress;
   else
      MemoryRequest->Size=0;

   if (MemoryRequest->RequiredSize>BlockSize-WordSize)
      MemoryRequest->RequiredSize=BlockSize-WordSize;
   else if (MemoryRequest->RequiredSize>MemoryRequest->Address-OldAddress)
      MemoryRequest->RequiredSize-=MemoryRequest->Address-OldAddress;
   else
      MemoryRequest->RequiredSize=0;

   if (BlockAddress(OldAddress)<BlockAddress(MemoryRequest->Address))
      MemoryRequest->Address-=BlockSize;

   }
```

```
/*****************************************************************************
**                                                                Page  6- 0  **
**                              TimeEst.c                                    **
**                                                                           **
**                           Part Of SACS 1.0                                **
**                     (StillAnother Cache Simulator)                        **
**                                                                           **
**   Program Modified: 3/17/94                                               **
**   File Modified:    3/17/94                                               **
**                                                                           **
**   Author:  William G. Smith                                               **
**   Address:  Electrical Engineering Department                             **
**             Naval Postgraduate School                                     **
**             Monterey, CA 93940                                            **
**                                                                           **
**   Copyright 1994, William G. Smith                                        **
**                                                                           **
**   Permission to use, copy, modify, and distribute this software and       **
**   its documentation for any purpose and without fee is hereby granted     **
**   provided that the above copyright notice appears in all copies.  No     **
**   modified version of this program should be redistributed without the    **
**   authors consent.  William G. Smith makes no warranty or                 **
**   representation, promise of guarantee, either expressed or implied,      **
**   with respect to this software's ability to produce valid results.       **
**   This program is provided "as is" any financial, personal or property    **
**   damage caused by the use of this program is the responsibility of the   **
**   user.                                                                   **
**                                                                           **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                              Page  6- 1  **
**                              TimeEst.c                                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       TimeEst.c contains all functions that relate to estimating the     **
**  execution, and completion times of memory requests.                    **
**                                                                          **
**                            Table of Contents                             **
**                                                                          **
**          Cover Page ............................... Page  6- 1          **
**          List of TimeEst.c Function Declarations ... Page  6- 2          **
**          UpdateTimeToExecute() ..................... Page  6- 3          **
**          CalculateTimeEstimates() .................. Page  6- 5          **
**                                                                          **
*****************************************************************************/

#include "Global.h"
```

```
/*****************************************************************************
**                                                              Page  6- 2  **
**                                TimeEst.c                                **
**                                                                         **
**                 List of TimeEst.c Function Declarations                 **
**                                                                         **
**  Description:                                                           **
**                                                                         **
**       This is a list of function declarations within the file scope     **
**  of TimeEst.c                                                           **
**                                                                         **
*****************************************************************************/


void UpdateTimeToExecute();                                  /* Page  6- 3 */
void CalculateTimeEstimates();                               /* Page  6- 5 */
```

```
/*****************************************************************************
**                                                          Page  6- 3  **
**                                TimeEst.c                              **
**                                                                       **
**                           UpdateTimeToExecute                         **
**                                                                       **
**  Description:                                                         **
**                                                                       **
**       UpdateTimeToExecute calculates the time to complete a memory    **
**  transfer given the MemoryRequest.  The Memory Request could be a read **
**  or write request in a buffer.  UpdateTimeToExecute changes the       **
**  TimeToExecute field to the new value.  TimeToexecute is calculated by **
**  first finding the number of WordsToBeTransfered.  If the MemoryRequest **
**  is not being accessed then the TimeToExecute is simply the AccessTime **
**  plus the TransferTime times one less then WordsToBeWritten.  If the  **
**  MemoryRequest is in progress then the new TimeToExecute is dependent **
**  on TOA, or TOD of the next word.  MemoryWaitingFor dictates whether to **
**  use the TOA, or TOD.  If MemoryWaitingFor is equal to CacheUpdate then **
**  the request has not actually begun transferring data.  So the        **
**  TimeToExecute can be calculated as if the read request is not in     **
**  progress.                                                            **
**                                                                       **
*****************************************************************************/


void UpdateTimeToExecute(MemoryRequest)

   MemoryRequestType *MemoryRequest;

   {

   SizeType WordsToBeTransfered;

   if (MemoryRequest->Size>0)
      {
      WordsToBeTransfered=WordAddress(MemoryRequest->Address
                                      +MemoryRequest->Size-1)
                          -WordAddress(MemoryRequest->Address)+WordSize;
      }
   else
      {
      WordsToBeTransfered=0;
      }

   WordsToBeTransfered/=WordSize;

   if (WordsToBeTransfered>(BlockSize/WordSize))
      WordsToBeTransfered=BlockSize/WordSize;
```

```
/*****************************************************************************
**                                                                          **
**                                TimeEst.c                                 **
**                                                                          **
**                           UpdateTimeToExecute                            **
**                                continued                                 **
**                                                                          **
*****************************************************************************/


   if (WordsToBeTransfered>0)
      {
      if (MemoryRequest->AccessInProgress==No)
         {
         MemoryRequest->TimeToExecute=MemoryAccessTime
                           +MemoryTransferTime*(WordsToBeTransfered-1);
         }
      else
         {
         if (MemoryWaitingFor==MemoryWaitingForMemoryReadAccess ||
             MemoryWaitingFor==MemoryWaitingForMemoryReadTransfer)
            {
            MemoryRequest->TimeToExecute=TOA-Time+MemoryTransferTime
                                        *(WordsToBeTransfered-1);
            }
         else if (MemoryWaitingFor==MemoryWaitingForMemoryWriteAccess ||
                  MemoryWaitingFor==MemoryWaitingForMemoryWriteTransfer)
            {
            MemoryRequest->TimeToExecute=TOD-Time+MemoryTransferTime
                                        *(WordsToBeTransfered-1);
            }
         else if (MemoryWaitingFor==MemoryWaitingForCacheUpdate)
            {
            MemoryRequest->TimeToExecute=MemoryAccessTime+MemoryTransferTime
                                        *(WordsToBeTransfered-1);
            }
         else
            {
            printf("Error found in [UpdateTimeToExecute] MemoryRequest\n");
            printf("with access in progress while MemoryWaitingFor not\n");
            printf("reading or writing.");
            DiscrepancyFound=Yes;
            }

         }
      }
   else
      {
      MemoryRequest->TimeToExecute=0;
      }

   }
```

```
/*****************************************************************************
**                                                          Page  6- 5  **
**                                TimeEst.c                              **
**                                                                       **
**                         CalculateTimeEstimates                        **
**                                                                       **
**  Description:                                                         **
**                                                                       **
**       CalculateTimeEstimates updates the CompletionTimeEstimates for  **
**  each request in both the read and write buffers.  This funtion is   **
**  called when ever the CacheModel adds to the read or write buffers.  **
**  CalculateTimeEstimates must be called every time new data is entered **
**  into the buffers.  This is because all previous estimates did not take **
**  into account the new data requested.  This is because all previous  **
**  estimates did not take into account the new data requested.         **
**  CalculateTimeEstimates first orders all entries in both the ReadBuffer, **
**  and the WriteBuffer by priority.  Then CalculateTimeEstimates steps  **
**  though both buffers simultaneously.  Each time picking the request that **
**  has the highest priority, and adding the time to execute to the     **
**  TimeEstimate.  The TimeEstimate becomes that requests               **
**  CompletionTimeEstimate.  This process is repeated until all requests **
**  have a new CompletionTimeEstimate.  TimeToExecute for cache request is **
**  updated before it is used to calculate the TimeEstimate.            **        **
**                                                                       **
*****************************************************************************/


void CalculateTimeEstimates()

   {

   BufferSizeType ReadIndex=0;
   BufferSizeType WriteIndex=0;
   TimeType       TimeEstimate=Time;
   TimeType       BlockTOAEstimate=BlockTOA;

   Order(&ReadBuffer);
   Order(&WriteBuffer);
```

```
/*****************************************************************************
**                                                                Page  6- 6  **
**                                TimeEst.c                                  **
**                                                                           **
**                          CalculateTimeEstimates                           **
**                                continued                                  **
**                                                                           **
*****************************************************************************/


   while (ReadIndex<ReadBuffer.Next || WriteIndex<WriteBuffer.Next)

      {
      if (ReadIndex<ReadBuffer.Next && WriteIndex<WriteBuffer.Next)
         {
         if (ReadBuffer.MemoryRequest[ ReadIndex ].Priority <=
             WriteBuffer.MemoryRequest[WriteIndex].Priority)
            {
            UpdateTimeToExecute(&(ReadBuffer.MemoryRequest[ReadIndex]));

            if (TimeEstimate<BlockTOAEstimate) TimeEstimate=BlockTOAEstimate;
            TimeEstimate+=ReadBuffer.MemoryRequest[ReadIndex].TimeToExecute;
            ReadBuffer.MemoryRequest[ReadIndex].CompletionTimeEstimate=
               TimeEstimate;
            BlockTOAEstimate=TimeEstimate+BufferCacheAccessTime;
            ReadIndex++;
            }
         else
            {
            UpdateTimeToExecute(&(WriteBuffer.MemoryRequest[WriteIndex]));

            TimeEstimate+=WriteBuffer.MemoryRequest[WriteIndex].TimeToExecute;
            WriteBuffer.MemoryRequest[WriteIndex].CompletionTimeEstimate=
               TimeEstimate;
            WriteIndex++;
            }
         }
      else if (ReadIndex<ReadBuffer.Next)
         {
         UpdateTimeToExecute(&(ReadBuffer.MemoryRequest[ReadIndex]));

         if (TimeEstimate<BlockTOAEstimate) TimeEstimate=BlockTOAEstimate;
         TimeEstimate+=ReadBuffer.MemoryRequest[ReadIndex].TimeToExecute;
         ReadBuffer.MemoryRequest[ReadIndex].CompletionTimeEstimate=
            TimeEstimate;
         BlockTOAEstimate=TimeEstimate+BufferCacheAccessTime;
         ReadIndex++;
         }
      else if (WriteIndex<WriteBuffer.Next)
         {
         UpdateTimeToExecute(&(WriteBuffer.MemoryRequest[WriteIndex]));

         TimeEstimate+=WriteBuffer.MemoryRequest[WriteIndex].TimeToExecute;
         WriteBuffer.MemoryRequest[WriteIndex].CompletionTimeEstimate=
            TimeEstimate;
         WriteIndex++;
         }

      }

   }
```

```
/*****************************************************************************
**                                                          Page  7- 0      **
**                                Get.c                                     **
**                                                                          **
**                          Part Of SACS 1.0                                **
**                    (StillAnother Cache Simulator)                        **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                              Page  7- 1  **
**                                 Get.c                                    **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Get.c contains all functions that relate getting the next CPU      **
**  request.  GetNextRequest is the only procedure called outside of this   **
**  file scope.  It determines whether to take input from the keyboard or   **
**  an input file.  It also checks the input data to see if it makes sense. **
**                                                                          **
**                             Table of Contents                            **
**                                                                          **
**          Cover Page ............................... Page  7- 1          **
**          List of Get.c Function Declarations ....... Page  7- 2          **
**          GetNextRequest() .......................... Page  7- 3          **
**             GetNextFileRequest() ................... Page  7- 5          **
**             GetNextKeyBoardRequest() ............... Page  7- 6          **
**                                                                          **
*****************************************************************************/


#include "Global.h"
```

```
/*****************************************************************************
**                                                              Page  7- 2  **
**                                  Get.c                                   **
**                                                                          **
**                   List of Get.c Function Declarations                    **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "Get.c".                                                             **
**                                                                          **
*****************************************************************************/


void GetNextRequest();                                        /* Page  7- 3 */
void    GetNextFileRequest();                                 /* Page  7- 5 */
void    GetNextKeyBoardRequest();                             /* Page  7- 6 */
```

```
/*****************************************************************************
**                                                          Page  7- 3   **
**                                Get.c                                   **
**                                                                        **
**                          GetNextRequest                                **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       GetNextRequest gets the next simulated request from the CPU to   **
**  cache (ie a Read or Write request).  The request is checked to make   **
**  sure it makes sense.  If a request is not block alined, then          **
**  GetNextRequest will split the request up and return portions of the   **
**  request until all portions have been used, as if the user had made    **
**  several different requests.                                           **
**                                                                        **
*****************************************************************************/


void GetNextRequest()

   {

   static AddressType NextRequestAddress;
   static SizeType    NextRequestSize=0;
   static RequestType NextRequest;
```

```
/*****************************************************************************
**                                                                Page  7- 4  **
**                                 Get.c                                    **
**                                                                          **
**                            GetNextRequest                                **
**                              continued                                   **
**                                                                          **
*****************************************************************************/

   if (NextRequestSize>0)

      {
      if (NextRequestSize<=BlockSize)
         {
         RequestSize=NextRequestSize;
         NextRequestSize=0;
         }
      else
         {
         RequestSize=BlockSize;
         NextRequestSize-=BlockSize;
         }
      RequestAddress=BlockAddress(RequestAddress)+BlockSize;
      Request=NextRequest;
      }

   else

      {

      if (KeyBoardIO)
         {
         PauseForCommand();
         GetNextKeyBoardRequest();
         }
      else
         {
         GetNextFileRequest();
         }

      if (BlockAddress(RequestAddress)!=
          BlockAddress(RequestAddress+RequestSize-1))
         {
         NextRequestSize=RequestSize;
         RequestSize=(BlockAddress(RequestAddress)+BlockSize)-RequestAddress;
         NextRequestSize-=RequestSize;
         NextRequestAddress=BlockAddress(RequestAddress)+BlockSize;
         NextRequest=Request;
         }

      }

   if (Request!=None)
      {
      LastRequest=Request;
      NumberOfAccesses[Request]++;
      }

   }
```

```
/*****************************************************************************
**                                                          Page  7- 5   **
**                                Get.c                                   **
**                                                                        **
**                         GetNextFileRequest                             **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       GetNextFileRequest reads in one request, without doing any error **
**  checking.                                                             **
**                                                                        **
*****************************************************************************/


void GetNextFileRequest()

   {

   char RequestChar=' ',
        Chr;

   Request=None;

   if (feof(DataFile)) EndOfDataFile=Yes;

   while (RequestChar!='r' && RequestChar!='w' && RequestChar!='E' &&
          EndOfDataFile==No && !feof(DataFile))
      fscanf(DataFile,"%c",&RequestChar);

   if (feof(DataFile) || RequestChar=='E') EndOfDataFile=Yes;

   if (EndOfDataFile==No)
      {

      fscanf(DataFile,"%lX", &RequestAddress);
      fscanf(DataFile, "%u", &RequestSize);
      fscanf(DataFile, "%U", &TimeOfNextRequest);

      if (RequestChar=='r') Request=Read;
      if (RequestChar=='w') Request=Write;

      TimeOfNextRequest+=Time;

      while (RequestChar!='\n' && !feof(DataFile))
         fscanf(DataFile,"%c",&RequestChar);
      }

   if (feof(DataFile)) EndOfDataFile=Yes;

   }
```

```
/*****************************************************************************
**                                                          Page  7- 6  **
**                                Get.c                                  **
**                                                                       **
**                           GetNextBoardKey                             **
**                                                                       **
**  Description:                                                         **
**                                                                       **
**       GetNextFileRequest reads in one request, without doing any error **
**  checking.                                                            **
**                                                                       **
*****************************************************************************/


void GetNextKeyBoardRequest()

   {

   char chr;

   printf("Please enter request type (r,w). ");
   while (chr!='r' && chr!='w' && chr!='q')
     {
     scanf("%c", &chr);
     }
   if (chr=='q') exit(0);
   if (chr=='r') Request=Read; else Request=Write;
   printf("Please enter Address ");
   scanf("%U", &RequestAddress);
   printf("Please enter size ");
   scanf("%u", &RequestSize);
   printf("Time until next request. ");
   fflush(stdin);
   scanf("%U", &TimeOfNextRequest);

   TimeOfNextRequest+=Time;

   }
```

```
/*****************************************************************************
**                                                                Page  8- 0  **
**                              Display.c                                    **
**                                                                           **
**                           Part Of SACS 1.0                                **
**                     (StillAnother Cache Simulator)                        **
**                                                                           **
**   Program Modified: 3/17/94                                               **
**   File Modified:    3/17/94                                               **
**                                                                           **
**   Author:  William G. Smith                                               **
**   Address:  Electrical Engineering Department                             **
**             Naval Postgraduate School                                     **
**             Monterey, CA 93940                                            **
**                                                                           **
**   Copyright 1994, William G. Smith                                        **
**                                                                           **
**   Permission to use, copy, modify, and distribute this software and       **
**   its documentation for any purpose and without fee is hereby granted     **
**   provided that the above copyright notice appears in all copies.  No     **
**   modified version of this program should be redistributed without the    **
**   authors consent.  William G. Smith makes no warranty or                 **
**   representation, promise of guarantee, either expressed or implied,      **
**   with respect to this software's ability to produce valid results.       **
**   This program is provided "as is" any financial, personal or property    **
**   damage caused by the use of this program is the responsibility of the   **
**   user.                                                                   **
**                                                                           **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                                Page  8- 1  **
**                              Display.c                                    **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**      Display.c contains all display functions used within SACS.           **
**                                                                           **
**                            Table of Contents                              **
**                                                                           **
**         Cover Page ............................... Page  8- 1            **
**         List of Cache.c Function Declarations ..... Page  8- 2            **
**         DisplayTrace() ........................... Page  8- 3            **
**            DisplayCurrentRequest() ............... Page  8- 4            **
**            DisplayWaitingFors() .................. Page  8- 5            **
**            DisplayBlock() ........................ Page  8- 6            **
**            DisplayBuffers() ...................... Page  8- 7            **
**               DisplayBuffer() .................... Page  8- 8            **
**                                                                           **
**         DisplayRequestsBreakDown() ............... Page  8- 9            **
**            DisplayRequestHistogram() ............. Page  8-11            **
**                                                                           **
**         DisplayStallHistogram() .................. Page  8-13            **
**            LastScreenHistogramScore() ............ Page  8-14            **
**         DisplayCacheArguments() .................. Page  8-15            **
**         DisplayHelp() ............................ Page  8-17            **
**                                                                           **
**         DisplayTestingHeader() ................... Page  8-18            **
**                                                                           **
**         PrintTime() .............................. Page  8-20            **
**         PrintTimeCentered() ...................... Page  8-20            **
**         PrintScoreCentered() ..................... Page  8-20            **
**         PrintAddress() ........................... Page  8-20            **
**         PrintCacheSize() ......................... Page  8-20            **
**         PrintSize() .............................. Page  8-20            **
**         PrintSize2() ............................. Page  8-20            **
**         PrintBufferSize() ........................ Page  8-21            **
**         PrintPriority() .......................... Page  8-21            **
**         PrintAssociativity() ..................... Page  8-21            **
**         PrintHistogramIndex() .................... Page  8-21            **
**                                                                           **
**         PrintBit() ............................... Page  8-22            **
**         PrintPercent() ........................... Page  8-22            **
**         PrintAveAccess() ......................... Page  8-22            **
**                                                                           **
*****************************************************************************/

#include "Global.h"
```

```
/*****************************************************************************
**                                                                Page  8- 2  **
**                              Display.c                                    **
**                                                                           **
**                List of Display.c Function Declarations                    **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       This is a list of function declarations within the file scope       **
**  of "Display.c".                                                          **
**                                                                           **
*****************************************************************************/


void      DisplayTrace();                                  /* Page  8- 3 */
void         DisplayCurrentRequest();                      /* Page  8- 4 */
void         DisplayWaitingFors();                         /* Page  8- 5 */
void         DisplayBlock();                               /* Page  8- 6 */
void         DisplayBuffers();                             /* Page  8- 7 */
void            DisplayBuffer();                           /* Page  8- 8 */

void      DisplayRequestsBreakDown();                      /* Page  8- 9 */
void         DisplayRequestHistogram();                    /* Page  8-11 */

void      DisplayStallHistogram();                         /* Page  8-13 */
ScoreType    LastScreenHistogramScore();                   /* Page  8-14 */
void      DisplayCacheArguments();                         /* Page  8-15 */
void      DisplayHelp();                                   /* Page  8-17 */

void      DisplayTestingHeader();                          /* Page  8-18 */

void      PrintYesNo();                                    /* Page  8-19 */
void      PrintRequest();                                  /* Page  8-19 */
void      PrintReplacementPolicy();                        /* Page  8-19 */
void      PrintWritePolicy();                              /* Page  8-19 */
void      PrintWriteMissPolicy();                          /* Page  8-19 */
void      PrintWaitingFor();                               /* Page  8-19 */
void      PrintMemoryWaitingFor();                         /* Page  8-19 */
void      PrintBlockWaitingFor();                          /* Page  8-19 */

void      PrintTime();                                     /* Page  8-20 */
void      PrintTimeCentered();                             /* Page  8-20 */
void      PrintScoreCentered();                            /* Page  8-20 */
void      PrintAddress();                                  /* Page  8-20 */
void      PrintCacheSize();                                /* Page  8-20 */
void      PrintSize();                                     /* Page  8-20 */
void      PrintSize2();                                    /* Page  8-20 */
void      PrintBufferSize();                               /* Page  8-21 */
void      PrintPriority();                                 /* Page  8-21 */
void      PrintAssociativity();                            /* Page  8-21 */
void      PrintHistogramIndex();                           /* Page  8-21 */

void      PrintBit();                                      /* Page  8-22 */
void      PrintPercent();                                  /* Page  8-22 */
void      PrintAveAccess();                                /* Page  8-22 */
```

```
/*****************************************************************************
**                                                          Page  8- 3  **
**                              Display.c                                **
**                                                                       **
**                             DisplayTrace                              **
**                                                                       **
*****************************************************************************/


void DisplayTrace()

   {

   SizeType Block0=Set(RequestAddress)*Associativity;
   SizeType BlockIndex;
   SizeType SubBlockIndex;

   system(ClearScreen);

   DisplayCurrentRequest();
   DisplayWaitingFors();

   printf("\n");
   printf("  Set Block  Address ");
   for (SubBlockIndex=0; SubBlockIndex<NumberOfSubBlocks; SubBlockIndex++)
      printf(" V/D ");
   printf("\n");

   for (BlockIndex=Block0; BlockIndex<Block0+Associativity; BlockIndex++)
      DisplayBlock(BlockIndex);

   printf("\n");

   DisplayBuffers();

   }
```

```
/*****************************************************************************
**                                                          Page  8- 4  **
**                              Display.c                                **
**                                                                       **
**                        DisplayCurrentRequest                          **
**                                                                       **
*****************************************************************************/


void DisplayCurrentRequest()

   {

   if (Request!=None) LastRequest=Request;

   if (LastRequest==None)
      {
      RequestAddress=0;
      RequestSize=0;
      CacheHit=No;
      }

   if (Request!=None) LastRequest=Request;
   if (CacheWaitingFor!=Nothing)
      {
      printf("\nCurrent Request:    ");
      }
   else
      {
      printf("\nLast Request:       ");
      }

   PrintRequest(LastRequest);
   printf("                    Time:                ");
   PrintTime(Time);

   printf("\nAddress:            ");
   PrintAddress(RequestAddress);
   printf("                  Next Request Time:  ");
   PrintTime(TimeOfNextRequest);

   printf("\nSize:               ");
   PrintSize2(RequestSize);

   if (MemoryWaitingFor==MemoryWaitingForMemoryReadAccess ||
       MemoryWaitingFor==MemoryWaitingForMemoryReadTransfer)
      {
      printf("                        TOA:                ");
      PrintTime(TOA);
      }
   if (MemoryWaitingFor==MemoryWaitingForMemoryWriteAccess ||
       MemoryWaitingFor==MemoryWaitingForMemoryWriteTransfer)
      {
      printf("                        TOD:                ");
      PrintTime(TOD);
      }

   printf("\n");

   }
```

```
/*****************************************************************************
**                                                              Page  8- 5  **
**                                Display.c                                 **
**                                                                          **
**                            DisplayWaitingFors                            **
**                                                                          **
*****************************************************************************/


void DisplayWaitingFors()

   {

   printf(  "Cache Waiting for:  "); PrintWaitingFor(CacheWaitingFor);

   printf("\nMemory Waiting For: "); PrintMemoryWaitingFor(MemoryWaitingFor);
   printf("      Cache Hit:          ");   PrintYesNo(CacheHit);
   printf("\nBlock Waiting For:  "); PrintBlockWaitingFor(BlockWaitingFor);
   printf("      Buffer Hit:         "); PrintYesNo(BufferHit);

   printf("\n");

   }
```

```
/*****************************************************************************
**                                                              Page  8- 6  **
**                                Display.c                                 **
**                                                                          **
**                               DisplayBlock                               **
**                                                                          **
*****************************************************************************/


void DisplayBlock(BlockIndex)

   SizeType BlockIndex;

   {

   SizeType SubBlockIndex;

   if (BlockIndex%Associativity==0)
      {
      PrintSize(BlockIndex/Associativity);
      }
   else
      {
      printf("     ");
      }
   printf(" ");

   PrintSize(BlockIndex);
   printf("  ");
   PrintAddress(CacheBlockAddress[BlockIndex]);

   for (SubBlockIndex=0; SubBlockIndex<NumberOfSubBlocks; SubBlockIndex++)
      {
      printf(" ");
      PrintBit(CacheValidBit[BlockIndex][SubBlockIndex]);
      printf(" ");
      PrintBit(CacheDirtyBit[BlockIndex][SubBlockIndex]);
      printf(" ");
      }

   printf("\n");

   }
```

```
/*****************************************************************************
**                                                              Page  8- 7  **
**                                Display.c                                 **
**                                                                          **
**                              DisplayBuffers                              **
**                                                                          **
*****************************************************************************/


void DisplayBuffers()

   {

   printf("Read Buffer  ");
   DisplayBuffer(&ReadBuffer);

   printf("\n");
   printf("Write Buffer ");
   DisplayBuffer(&WriteBuffer);

   printf("\n");
   printf("Block Buffer ");
   DisplayBuffer(&BlockBuffer);

   }
```

```
/*****************************************************************************
**                                                              Page  8- 8  **
**                                Display.c                                 **
**                                                                          **
**                              DisplayBuffer                               **
**                                                                          **
*****************************************************************************/


void DisplayBuffer(PrintBuffer)

   BufferType *PrintBuffer;

   {

   int R;

   printf("Address    Size   Req.   Block   Priority");
   printf("   Time Req.   Comp. Time\n");

   for (R=0; R<PrintBuffer->Next; R++)
      {
      printf("              ");
      PrintAddress(PrintBuffer->MemoryRequest[R].Address);
      printf("    ");
      PrintSize2(PrintBuffer->MemoryRequest[R].Size);
      printf("     ");
      PrintSize2(PrintBuffer->MemoryRequest[R].RequiredSize);
      printf("    ");
      PrintSize(PrintBuffer->MemoryRequest[R].Block);
      printf("      ");
      PrintPriority(PrintBuffer->MemoryRequest[R].Priority);
      printf("      ");
      PrintTimeCentered(PrintBuffer->MemoryRequest[R].TimeToExecute);
      printf("     ");
      PrintTimeCentered(PrintBuffer->MemoryRequest[R].CompletionTimeEstimate);
      printf("\n");
      }

   }
```

```
/*****************************************************************************
**                                                              Page  8- 9  **
**                                Display.c                                 **
**                                                                          **
**                          DisplayRequestBreakDown                         **
**                                                                          **
*****************************************************************************/


void DisplayRequestsBreakDown()

   {

   ScoreType
      TotalNumberOfAccesses =NumberOfAccesses[Read] +NumberOfAccesses[Write],
      TotalNumberOfCacheHits=NumberOfCacheHits[Read]+NumberOfCacheHits[Write],
      TotalNumberOfBufferHits=NumberOfBufferHits[Read]+NumberOfBufferHits[Write];
   system(ClearScreen);

   printf("\n                                Requests Break Down\n");
   printf("\n      ");
   printf("              Number       Number      Number                      ");
   printf("\n      ");
   printf("Request         of           of          of           Hit      Miss ");
   printf("\n      ");
   printf(" Types       Requests    Cache Hits  Buffer Hits     Rates     Rates");

   printf("\n");
   printf("\n       Read        ");
   PrintScoreCentered(NumberOfAccesses[Read]);
   printf("     ");
   PrintScoreCentered(NumberOfCacheHits[Read]);
   printf("    ");
   PrintScoreCentered(NumberOfBufferHits[Read]);
   if (NumberOfAccesses[Read]>0)
      {
      printf("      ");
      PrintPercent(NumberOfCacheHits[Read]+NumberOfBufferHits[Read],
                   NumberOfAccesses[Read]);
      printf("   ");
      PrintPercent((NumberOfAccesses[Read]-NumberOfCacheHits[Read]),
                    NumberOfAccesses[Read]);
      }

   printf("\n       Write       ");
   PrintScoreCentered(NumberOfAccesses[Write]);
   printf("     ");
   PrintScoreCentered(NumberOfCacheHits[Write]);
   printf("    ");
   PrintScoreCentered(NumberOfBufferHits[Write]);

   if (NumberOfAccesses[Write]>0)
      {
      printf("      ");
      PrintPercent(NumberOfCacheHits[Write]+NumberOfBufferHits[Write],
                   NumberOfAccesses[Write]);
      printf("   ");
      PrintPercent((NumberOfAccesses[Write]-NumberOfCacheHits[Write]),
                    NumberOfAccesses[Write]);
      }
```

```
/*****************************************************************************
**                                                                          **
**                                Display.c                                 **
**                                                                          **
**                          DisplayRequestBreakDown                         **
**                                continued                                 **
**                                                                          **
*****************************************************************************/


   printf("\n       Total       ");
   PrintScoreCentered(TotalNumberOfAccesses);
   printf("     ");
   PrintScoreCentered(TotalNumberOfCacheHits);
   printf("    ");
   PrintScoreCentered(TotalNumberOfBufferHits);

   if (TotalNumberOfAccesses>0)
      {
      printf("      ");
      PrintPercent(TotalNumberOfCacheHits+TotalNumberOfBufferHits,
                   TotalNumberOfAccesses);
      printf("   ");
      PrintPercent(TotalNumberOfAccesses-TotalNumberOfCacheHits
                                        -TotalNumberOfBufferHits,
                   TotalNumberOfAccesses);
      }

   printf("\n");

   DisplayRequestHistogram();

   }
```

```
/*****************************************************************************
**                                                                Page  8-11  **
**                                Display.c                                  **
**                                                                           **
**                          DisplayRequestHistogram                          **
**                                                                           **
*****************************************************************************/


void DisplayRequestHistogram()

   {

   SizeType TimeIndex;

   printf("\n                                Request Time Histogram");
   printf("\n              ");
   for (TimeIndex=0; TimeIndex<ScreenHistogramMaxIndex; TimeIndex++)
      printf("          ");

   printf("             Ave  ");
   printf("\n              ");
   for (TimeIndex=0; TimeIndex<ScreenHistogramMaxIndex; TimeIndex++)
      printf("          ");

   printf("            Access");
   printf("\n             ");

   for (TimeIndex=0; TimeIndex<ScreenHistogramMaxIndex-1; TimeIndex++)
      {
      printf("  Time=");
      PrintSize2(TimeIndex);
      }
   printf("  Time>=");
   PrintSize2(TimeIndex);
   printf("   Total    Time ");
```

```
   printf("\n       Read  ");
   for (TimeIndex=0; TimeIndex<(ScreenHistogramMaxIndex-1); TimeIndex++)
      {
      printf(" ");
      PrintScoreCentered(RequestTimeHistogram[Read][TimeIndex]);
      }
   printf(" ");
   PrintScoreCentered(LastScreenHistogramScore(RequestTimeHistogram[Read]));
   printf(" ");
   PrintScoreCentered(TotalRequestTime[Read]);
   printf(" ");
   PrintAveAccess(TotalRequestTime[Read],NumberOfAccesses[Read]);

   printf("\n       Write ");
   for (TimeIndex=0; TimeIndex<(ScreenHistogramMaxIndex-1); TimeIndex++)
      {
      printf(" ");
      PrintScoreCentered(RequestTimeHistogram[Write][TimeIndex]);
      }
   printf(" ");
   PrintScoreCentered(LastScreenHistogramScore(RequestTimeHistogram[Write]));
   printf(" ");
   PrintScoreCentered(TotalRequestTime[Write]);
   printf(" ");
   PrintAveAccess(TotalRequestTime[Write],NumberOfAccesses[Write]);

   printf("\n       Ideal ");
   for (TimeIndex=0; TimeIndex<(ScreenHistogramMaxIndex-1); TimeIndex++)
      {
      printf(" ");
      PrintScoreCentered(RequestTimeHistogram[None][TimeIndex]);
      }
   printf(" ");
   PrintScoreCentered(LastScreenHistogramScore(RequestTimeHistogram[None]));
   printf(" ");
   PrintScoreCentered(TotalRequestTime[None]);
   printf("\n");

   }
```

```
/*****************************************************************************
**                                                                Page  8-13  **
**                                Display.c                                  **
**                                                                           **
**                          DisplayStallHistogram                            **
**                                                                           **
*****************************************************************************/


void DisplayStallHistogram()

   {

   SizeType            TimeIndex;
   CacheWaitingForType StallIndex;

   system(ClearScreen);

   printf("\n\n                              Stall Time Histogram\n\n");
   printf("                    ");
   for (TimeIndex=0;TimeIndex<ScreenHistogramMaxIndex-1;TimeIndex++)
      {
      printf("  Time=");
      PrintSize2(TimeIndex);
      }
   printf("  Time>=");
   PrintSize2(TimeIndex);
   printf("   Total\n");
   printf("\n");

   for (StallIndex=0;
        StallIndex<NumberOfCacheWaitingForsAvailable;
        StallIndex++)
      {
      PrintWaitingFor(StallIndex);
      for (TimeIndex=0;TimeIndex<ScreenHistogramMaxIndex-1;TimeIndex++)
         {
         printf(" ");
         PrintScoreCentered(StallTimeHistogram[StallIndex][TimeIndex]);
         }

      printf(" ");
      PrintScoreCentered(LastScreenHistogramScore(StallTimeHistogram[StallIndex]));
      printf(" ");
      PrintScoreCentered(TotalStallTime[StallIndex]);
      printf("\n");
      }

   }
```

```
/*****************************************************************************
**                                                              Page  8-14  **
**                                Display.c                                 **
**                                                                          **
**                         LastScreenHistogramScore                         **
**                                                                          **
*****************************************************************************/


ScoreType LastScreenHistogramScore(Histogram)

   TimeType *Histogram;

   {

   TimeType  TimeIndex;
   ScoreType Sum=0;

   for (TimeIndex=ScreenHistogramMaxIndex-1;
        TimeIndex<FileHistogramMaxIndex;
        TimeIndex++)
      Sum+=Histogram[TimeIndex];

   return(Sum);

   }
```

```
/*****************************************************************************
**                                                          Page  8-15  **
**                              Display.c                                  **
**                                                                         **
**                        DisplayCacheArguments                            **
**                                                                         **
*****************************************************************************/


void DisplayCacheArguments()

   {

   system(ClearScreen);

   printf("\n");
   printf("                              Cache Arguments List");
   printf("\n");
   printf("\nCache Size:                 "); PrintCacheSize(CacheSize);
   printf("       ");
   printf("Read Forward:                 "); PrintYesNo(ReadForward);
   printf("\nBlock Size:                 "); PrintSize(BlockSize);
   printf("          ");
   printf("CPU Waits For Cache Writes: "); PrintYesNo(CPUWaitsForCacheWrites);
   printf("\nSubBlock Size:              "); PrintSize(SubBlockSize);
   printf("          ");
   printf("Search Block Buffer:          "); PrintYesNo(SearchBlockBuffer);
   printf("\nAssociativity:              "); PrintAssociativity(Associativity);
   printf("             ");
   printf("Update Read Buffer:           "); PrintYesNo(UpdateReadBuffer);
   printf("\nWord Size:                  "); PrintSize(WordSize);
   printf("          ");
   printf("Remove Read Duplicates:       "); PrintYesNo(RemoveReadDuplicates);
   printf("\nRead Cache Access Time:     "); PrintTime(ReadCacheAccessTime);
   printf("       ");
   printf("Remove Write Duplicates:      "); PrintYesNo(RemoveWriteDuplicates);
   printf("\nRead Cache Hit Time:        "); PrintTime(ReadCacheHitTime);
   printf("       ");
   printf("Read Priority:                "); PrintPriority(ReadPriority);
   printf("\nRead Cache Miss Time:       "); PrintTime(ReadCacheMissTime);
   printf("       ");
   printf("Write Priority:               "); PrintPriority(WritePriority);
   printf("\nWrite Cache Access Time:    "); PrintTime(WriteCacheAccessTime);
   printf("       ");
   printf("Read For Write Allocate:      ");
      PrintPriority(ReadForWriteAllocatePriority);
   printf("\nWrite Cache Hit Time:       "); PrintTime(WriteCacheHitTime);
   printf("       ");
   printf("Write Dirty Block Priority: ");
      PrintPriority(WriteDirtyBlockPriority);
   printf("\nWrite Cache Miss Time:      "); PrintTime(WriteCacheMissTime);
   printf("       ");
   printf("No Priority:                  "); PrintPriority(NoPriority);
```

Display.c

DisplayCacheArguments
continued

```
   printf("\nMemory Access Time:       "); PrintTime(MemoryAccessTime);
   printf("       ");
   printf("Trace:                      "); PrintYesNo(Trace);
   printf("\nMemory Transfer Time:     "); PrintTime(MemoryTransferTime);
   printf("       ");
   printf("Check:                      "); PrintYesNo(Check);
   printf("\nBuffer Cache Access Time: "); PrintTime(BufferCacheAccessTime);
   printf("       ");
   printf("Test:                       "); PrintYesNo(Test);
   printf("\nRead Buffer Size:         "); PrintBufferSize(ReadBufferSize);
   printf("              ");
   printf("Key Board IO:               "); PrintYesNo(KeyBoardIO);
   printf("\nWrite Buffer Size:        "); PrintBufferSize(WriteBufferSize);
   printf("              ");
   printf("Data File Name:             %s",DataFileName);
   printf("\nBlock Replacement Policy: ");
      PrintReplacementPolicy(BlockReplacementPolicy);
   printf("            ");
   printf("Screen History Max Index:   ");
      PrintHistogramIndex(ScreenHistogramMaxIndex);
   printf("\nWrite Policy              "); PrintWritePolicy(WritePolicy);
   printf("   ");
   printf("File History Max Index:     ");
      PrintHistogramIndex(FileHistogramMaxIndex);
   printf("\nWrite Miss Policy:        ");
      PrintWriteMissPolicy(WriteMissPolicy);

   printf("\n");

   }
```

```
/*****************************************************************************
**                                                              Page  8-17  **
**                                Display.c                                 **
**                                                                          **
**                               DisplayHelp                                **
**                                                                          **
*****************************************************************************/


void DisplayHelp()

   {

   system(ClearScreen);

   printf("                          Help Menu                            ");

   printf("\n");
   printf("\n [T] Trace Display:                                           ");
   printf("\n     Displays current request, status of memory, and contents ");
   printf("\n     of buffers.                                              ");

   printf("\n");
   printf("\n [R] Results Display:                                         ");
   printf("\n     Displays a break down of read and write cache hits, and  ");
   printf("\n     buffer hits, including a timing analysis.                ");

   printf("\n");
   printf("\n [S] Stall Timing Display:                                    ");
   printf("\n     Displays a histogram of the time spent on each stall.    ");
   printf("\n     Stalls represent time delays in completing a request     ");

   printf("\n");
   printf("\n [C] Cache Arguments Display:                                 ");
   printf("\n     Displays input arguments to SACS.                        ");

   printf("\n");
   printf("\n [G]   Go:        Go to end of run.                           ");
   printf("\n [G #] Go To:     Go to Time #.                               ");
   printf("\n [#]   Step:      Increment Time By #.                        ");
   printf("\n [-#]  Back Step: Decrement Time By #.                        ");
   printf("\n [H]   Help:      Displays this help menu.                    ");
   printf("\n                                                              ");

   }
```

```
/*****************************************************************************
**                                                                Page  8-18  **
**                              Display.c                                    **
**                                                                           **
**                         DisplayTestingHeader                              **
**                                                                           **
*****************************************************************************/


void DisplayTestingHeader()

   {

   printf("\n");
   system(ClearScreen);
   printf("\n\n");
   printf("\n\n                  ");
   printf("                 Testing SACS");
   printf("\n\n                  ");
   printf("Total number of loads and stores tested %lu.",
           TotalNumberOfAccesses);
   printf("\n\n                  ");
   printf("              Test Cases chosen ... ");
   printf("\n\n");

   }
```

```
/*****************************************************************************
**                                                          Page  8-19   **
**                              Display.c                                  **
**                                                                         **
**                        Print Enumeration Stings                         **
**                                                                         **
*****************************************************************************/


void PrintYesNo(Value)
   YesNoType Value;
   {
   printf("%s",YesNoString[Value]);
   }

void PrintRequest(Value)
   RequestType Value;
   {
   printf("%s",RequestString[Value]);
   }

void PrintReplacementPolicy(Value)
   BlockReplacementPolicyType Value;
   {
   printf("%s", ReplacementPolicyString[Value]);
   }

void PrintWritePolicy(Value)
   WritePolicyType Value;
   {
   printf("%s", WritePolicyString[Value]);
   }

void PrintWriteMissPolicy(Value)
   WriteMissPolicyType Value;
   {
   printf("%s",WriteMissPolicyString[Value]);
   }

void PrintWaitingFor(Value)
   CacheWaitingForType Value;
   {
   printf("%s",CacheWaitingForString[Value]);
   }

void PrintMemoryWaitingFor(Value)

   MemoryWaitingForType Value;

   {
   printf("%s",MemoryWaitingForString[Value]);
   }

void PrintBlockWaitingFor(Value)
   BlockWaitingForType Value;
   {
   printf("%s",BlockWaitingForString[Value]);
   }
```

```
/*****************************************************************************
**                                                                Page  8-20  **
**                                Display.c                                  **
**                                                                           **
**                             Print Routines                                **
**                                                                           **
*****************************************************************************/


void PrintTime(Time)
   TimeType Time;
   {
   if      (Time>=10000000) printf("%8lu",Time);
   else if (Time>=1000000 ) printf("%7lu ",Time);
   else if (Time>=100000  ) printf("%6lu  ",Time);
   else if (Time>=10000   ) printf("%5lu   ",Time);
   else if (Time>=1000    ) printf("%4lu    ",Time);
   else if (Time>=100     ) printf("%3lu     ",Time);
   else if (Time>=10      ) printf("%2lu      ",Time);
   else                     printf("%1lu       ",Time);
   }

void PrintTimeCentered(Time)
   TimeType Time;
   {
   if      (Time>=1000000) printf("%8lu",Time);
   else if (Time>=10000  ) printf("%7lu ",Time);
   else if (Time>=100    ) printf("%6lu  ",Time);
   else                    printf("%5lu   ",Time);
   }

void PrintScoreCentered(Score)
   ScoreType Score;
   {
   if      (Time>=1000000) printf("%8lu",Score);
   else if (Time>=10000  ) printf("%7lu ",Score);
   else if (Time>=100    ) printf("%6lu  ",Score);
   else                    printf("%5lu   ",Score);
   }

void PrintAddress(Address)
   AddressType Address;
   {
   printf("%08lX",Address);
   }

void PrintCacheSize(CacheSize)
   CacheSizeType CacheSize;
   {
   printf("%08lu",CacheSize);
   }

void PrintSize(Size)
   SizeType Size;
   {
   printf("%05u",Size);
   }

void PrintSize2(Size)
   SizeType Size;
   {
   printf("%02u",Size);
   }
```

```
/*****************************************************************************
**                                                                Page  8-21  **
**                                Display.c                                  **
**                                                                           **
**                             Print Routines                                **
**                               continued                                   **
**                                                                           **
*****************************************************************************/


void PrintBufferSize(BufferSize)
   BufferSizeType BufferSize;
   {
   printf("%02u",BufferSize);
   }

void PrintPriority(Priority)
   PriorityType Priority;
   {
   printf("%02u",Priority);
   }

void PrintAssociativity(Associativity)
   AssociativityType Associativity;
   {
   printf("%02u",Associativity);
   }

void PrintHistogramIndex(HistogramIndex)
   HistogramIndexType HistogramIndex;
   {
   printf("%04u",HistogramIndex);
   }
```

```
/*****************************************************************************
**                                                          Page  8-22   **
**                              Display.c                                 **
**                                                                        **
**                           Print Routines                               **
**                             continued                                  **
**                                                                        **
*****************************************************************************/


void PrintBit(Bit)
   YesNoType Bit;
   {
   printf("%01u",Bit);
   }

void PrintPercent(Numerator,Denominator)
   ScoreType Numerator;
   ScoreType Denominator;
   {
   if (Denominator>0)
      {
      printf("%6.2lf",(100.0*Numerator)/Denominator);
      printf("%");
      }
   else
      {
      printf("       ");
      }
   }

void PrintAveAccess(TotalTime,TotalNumberofAccesses)
   TimeType  TotalTime;
   ScoreType TotalNumberofAccesses;
   {
   if (TotalNumberofAccesses>0)
      {
      printf("%8.6lf",(1.0*TotalTime)/TotalNumberofAccesses);
      }
   else
      {
      printf("         ");
      }
   }
```

```
/*****************************************************************************
**                                                                         Page  9- 0  **
**                                  Record.c                                **
**                                                                          **
**                              Part Of SACS 1.0                            **
**                        (StillAnother Cache Simulator)                    **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                                Page  9- 1 **
**                                Record.c                                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**                                                                          **
**       Record.c contains all functions that relate to the recording of    **
**  time for requests, and waiting fors, as well as a procedure for saving  **
**  the data in a file using a format that Matlab(TM) could read.           **
**                                                                          **
**                             Table of Contents                            **
**                                                                          **
**          Cover Page ............................... Page  9- 1          **
**          List of Record.c Function Declarations .... Page  9- 2          **
**          RecordRequest() .......................... Page  9- 3          **
**          RecordStall() ............................ Page  9- 5          **
**          RecordForMatlab() ........................ Page  9- 7          **
**                                                                          **
*****************************************************************************/

#include "Global.h"
```

```
/*****************************************************************************
**                                                          Page  9- 2 **
**                              Record.c                                     **
**                                                                           **
**                List of Record.c Function Declarations                     **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       This is a list of function declarations within the file scope       **
**  of "Record.c".                                                           **
**                                                                           **
*****************************************************************************/


void  RecordRequest();                                    /* Page  9- 3 */
void  RecordStall();                                      /* Page  9- 5 */
void  RecordForMatlab();                                  /* Page  9- 7 */
```

```
/*****************************************************************************
**                                                                Page  9- 3 **
**                                Record.c                                  **
**                                                                          **
**                             Record Request                               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       RecordRequest records the time spent on a particular request and   **
**  stores the result in RequestTimeHistogram.                              **
**                                                                          **
*****************************************************************************/


void RecordRequest(Req)

   RequestType Req;

   {

   static TimeType    LastTime=1;
   static TimeType    Lastdt=0;
   static RequestType LastReq=NumberOfRequestsAvailable;
          TimeType    dt=Time-LastTime;

   if (Req==NumberOfRequestsAvailable)
      {
      LastTime=1;
      Lastdt=0;
      LastReq=Req;
      }
   else if (Req==LastReq)
      {
      TotalRequestTime[LastReq]-=Lastdt;
      if (Lastdt>FileHistogramMaxIndex-1) Lastdt=FileHistogramMaxIndex-1;
      RequestTimeHistogram[LastReq][Lastdt]--;
      }
   else if (LastReq!=NumberOfRequestsAvailable)
      {
      LastTime=Time;
      dt=0;
      NumberOfCacheHits[LastReq]+=CacheHit;
      NumberOfBufferHits[LastReq]+=BufferHit;
      }

   LastReq=Req;
```

```
/*****************************************************************************
**                                                                Page  9- 4 **
**                                Record.c                                  **
**                                                                          **
**                             Record Request                               **
**                               continued                                  **
**                                                                          **
*****************************************************************************/


   if (Req!=NumberOfRequestsAvailable)
      {
      TotalRequestTime[Req]+=dt;
      Lastdt=dt;
      if (dt>FileHistogramMaxIndex-1) dt=FileHistogramMaxIndex-1;
      if (Time>=LastTime)
         {
         RequestTimeHistogram[Req][dt]++;
         }
      else
         {
         printf("\n\nError [RecordRequest] caculated a time less than 0");
         printf("\n\n    Time = "); PrintTime(Time);
         printf("\n\nLastTime = "); PrintTime(LastTime);
         printf("\n\n");
         DiscrepancyFound=Yes;
         }
      }
   }
```

```
/*****************************************************************************
**                                                                Page  9- 5 **
**                                Record.c                                  **
**                                                                          **
**                               RecordStall                                **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       RecordStall records the time spent on a particular waiting for and **
**  stores the result in StallTimeHistogram.                                **
**                                                                          **
*****************************************************************************/


void RecordStall(CurrentWaitingFor)

   CacheWaitingForType CurrentWaitingFor;

   {

   static TimeType            PastTime=1;
   static TimeType            Pastdt=0;
   static CacheWaitingForType PastWaitingFor=None;
          TimeType            dt=Time-PastTime;

   if (CurrentWaitingFor==NumberOfCacheWaitingForsAvailable)
      {
      PastTime=1;
      Pastdt=0;
      PastWaitingFor=CurrentWaitingFor;
      }
   else if (CurrentWaitingFor==PastWaitingFor)
      {
      TotalStallTime[PastWaitingFor]-=Pastdt;
      if (Pastdt>FileHistogramMaxIndex-1) Pastdt=FileHistogramMaxIndex-1;
      StallTimeHistogram[PastWaitingFor][Pastdt]--;
      }
   else if (PastWaitingFor!=NumberOfCacheWaitingForsAvailable)
      {
      PastTime=Time;
      dt=0;
      }

   PastWaitingFor=CurrentWaitingFor;
```

```
/*****************************************************************************
**                                                                Page  9- 6 **
**                                Record.c                                  **
**                                                                          **
**                               RecordStall                                **
**                                continued                                 **
**                                                                          **
*****************************************************************************/


   if (CurrentWaitingFor!=NumberOfCacheWaitingForsAvailable)
      {
      TotalStallTime[CurrentWaitingFor]+=dt;
      Pastdt=dt;
      if (dt>Fi eHistogramMaxIndex-1) dt=FileHistogramMaxIndex-1;
      if (Time>=PastTime)
         {
         StallTimeHistogram[CurrentWaitingFor][dt]++;
         }
      else
         {
         printf("\n\nError [RecordStall] calculated a time less than 0");
         printf("\n\n    Time = "); PrintTime(Time);
         printf("\n\nPastTime = "); PrintTime(PastTime);
         printf("\n\n");
         DiscrepancyFound=Yes;
         }
      }
   }
```

```
/*****************************************************************************
**                                                                Page  9- 7 **
**                                Record.c                                  **
**                                                                          **
**                            RecordForMatlab                               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       RecordForMatlab saves the RequestTimeHistogram, and                **
**  StallTimeHistograms in a format that Matlab(TM) reconizes.              **
**                                                                          **
*****************************************************************************/


void RecordForMatlab()

   {

   CacheWaitingForType StallIndex;
   RequestType         RequestIndex;

   int                 Column,
                       NumberOfColumns=2;

   HistogramIndexType HistogramIndex;

   FILE *MatlabOut;

   if ((MatlabOut=fopen("timing.m","w"))==NULL)
      {
      printf("Can not open matlab output file.");
      }

   for (RequestIndex=0; RequestIndex<NumberOfRequestsAvailable; RequestIndex++)
      {

      fprintf(MatlabOut,"%s=[",RequestString[RequestIndex]);
      fprintf(MatlabOut," %08lu",RequestTimeHistogram[RequestIndex][0]);

      Column=1;

      for (HistogramIndex=1;
           HistogramIndex<FileHistogramMaxIndex;
           HistogramIndex++)
         {
         Column++;
         if (Column>NumberOfColumns)
            {
            Column=1;
            fprintf(MatlabOut,",\n       ");
            }
         else
            {
            fprintf(MatlabOut,",");
            }
         fprintf(MatlabOut," %08lu",
                 RequestTimeHistogram[RequestIndex][HistogramIndex]);
         }

      fprintf(MatlabOut,"];\n\n");

      }
```

```
/*****************************************************************************
**                                                                          **
**                                Record.c                                  **
**                                                                          **
**                             RecordForMatlab                              **
**                                Continued                                 **
**                                                                          **
*****************************************************************************/


   for (StallIndex=0;StallIndex<NumberOfCacheWaitingForsAvailable;StallIndex++)
      {
      fprintf(MatlabOut,"%s=[",CacheWaitingForString[StallIndex]);
      fprintf(MatlabOut," %08lu",StallTimeHistogram[StallIndex][0]);

      Column=1;

      for (HistogramIndex=1;
           HistogramIndex<FileHistogramMaxIndex;
           HistogramIndex++)
         {
         Column++;
         if (Column>NumberOfColumns)
            {
            Column=1;
            fprintf(MatlabOut,",\n                                  ");
            }
         else
            {
            fprintf(MatlabOut,",");
            }

         fprintf(MatlabOut," %08lu",
                 StallTimeHistogram[StallIndex][HistogramIndex]);
         }

      fprintf(MatlabOut,"];\n\n");

      }

   fclose(MatlabOut);

   }
```

```
/*****************************************************************************
**                                                          Page 10- 0      **
**                              Buffer.c                                    **
**                                                                          **
**                          Part Of SACS 1.0                                **
**                    (StillAnother Cache Simulator)                        **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                              Page 10- 1  **
**                                 Buffer.c                                 **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Buffer.c contains all functions that relate to the management of   **
**  the Read, Write, and Block Buffers.                                     **
**                                                                          **
**                              Table of Contents                           **
**                                                                          **
**          Cover Page ............................... Page 10- 1          **
**          List of Cache.c Function Declarations ..... Page 10- 2          **
**          Push() ................................... Page 10- 3          **
**          Pop() .................................... Page 10- 4          **
**          ChangeTopMemoryRequest() ................. Page 10- 5          **
**          Append() ................................. Page 10- 6          **
**          View() ................................... Page 10- 7          **
**          Clear() .................................. Page 10- 8          **
**          Order() .................................. Page 10- 9          **
**          Splice() ................................. Page 10-10          **
**          Search() ................................. Page 10-12          **
**          UpdatingReadBuffer() ..................... Page 10-13          **
**          RemoveZeroSizes() ........................ Page 10-15          **
**          NoRequestsLeft() ......................... Page 10-16          **
**                                                                          **
*****************************************************************************/

#include "Global.h"
```

```
/*****************************************************************************
**                                                              Page 10- 2  **
**                                Buffer.c                                  **
**                                                                          **
**                  List of Buffer.c Function Declarations                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of functions declarations within the file scope     **
**  of "Buffer.c".                                                          **
**                                                                          **
*****************************************************************************/


void              Push();                                  /* Page 10- 3 */
MemoryRequestType Pop();                                   /* Page 10- 4 */
void              ChangeTopMemoryRequest();                /* Page 10- 5 */
void              Append();                                /* Page 10- 6 */
MemoryRequestType View();                                  /* Page 10- 7 */
void              Clear();                                 /* Page 10- 8 */
void              Order();                                 /* Page 10- 9 */
void              Splice();                                /* Page 10-10 */
YesNoType         Search();                                /* Page 10-12 */
YesNoType         UpdatingReadBuffer();                    /* Page 10-13 */
void              RemoveZeroSizes();                       /* Page 10-15 */
YesNoType         NoRequestsLeft();                        /* Page 10-16 */
```

```
/*****************************************************************************
**                                                              Page 10- 3  **
**                                 Buffer.c                                **
**                                                                         **
**                                   Push                                  **
**                                                                         **
**  Description:                                                           **
**                                                                         **
**     Push adds a new record to the top of the buffer.                    **
**                                                                         **
*****************************************************************************/


void Push(Buffer, MemoryRequest)

   BufferType *Buffer;
   MemoryRequestType *MemoryRequest;

   {

   BufferSizeType i;

   if (Buffer->Full)
      {
      CacheWaitingFor=Buffer->WaitingForFlag;
      }
   else
      {
      for (i=Buffer->Next; i>0; i--)
         Buffer->MemoryRequest[i]=Buffer->MemoryRequest[i-1];
      Buffer->Next++;
      Buffer->MemoryRequest[0]=*MemoryRequest;
      if (Buffer->Next>Buffer->Max) Buffer->Full=Yes;
      Buffer->Empty=No;
      if (CacheWaitingFor==Buffer->WaitingForFlag) CacheWaitingFor=Nothing;
      }

   }
```

```
/*****************************************************************************
**                                                              Page 10- 4  **
**                                 Buffer.c                                 **
**                                                                          **
**                                   Pop                                    **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**     Pop removes a record from the top of the buffer, and returns it to   **
**  the caller of the function.                                             **
**                                                                          **
*****************************************************************************/


MemoryRequestType Pop(Buffer)

   BufferType *Buffer;

   {

   MemoryRequestType        MemoryRequest;
   BufferSizeType i;

   if (Buffer->Empty || Buffer->Next==0)
      {
      printf("\n\n   Tryed to Pop an empty buffer!\n\n");
      exit(1);
      }
   else
      {
      MemoryRequest=Buffer->MemoryRequest[0];
      for (i=0; i<Buffer->Next; i++)
         Buffer->MemoryRequest[i]=Buffer->MemoryRequest[i+1];
      Buffer->Next--;
      if (Buffer->Next==0) Buffer->Empty=Yes;
      Buffer->Full=No;
      }

   return(MemoryRequest);

   }
```

```
/*****************************************************************************
**                                                                Page 10- 5  **
**                                Buffer.c                                  **
**                                                                          **
**                         ChangeTopMemoryRequest                           **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**     Push adds a new record to the top of the buffer.                     **
**                                                                          **
*****************************************************************************/


void ChangeTopMemoryRequest(Buffer, MemoryRequest)

   BufferType *Buffer;
   MemoryRequestType *MemoryRequest;

   {

   if (Buffer->Empty==No && Buffer->Next>0)
      {
      Buffer->MemoryRequest[0]=*MemoryRequest;
      }
   else
      {
      printf("\n\n   Tryed to Pop an empty buffer!\n\n");
      exit(1);
      }

   }
```

```
/*****************************************************************************
**                                                          Page 10- 6   **
**                                Buffer.c                                **
**                                                                        **
**                                 Append                                 **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**     Append adds a new record to the bottom of the buffer.              **
**                                                                        **
*****************************************************************************/


void Append(Buffer, MemoryRequest)

   BufferType *Buffer;
   MemoryRequestType *MemoryRequest;

   {

   if (Buffer->Full)
      {
      CacheWaitingFor=Buffer->WaitingForFlag;
      }
   else
      {
      Buffer->MemoryRequest[Buffer->Next]=*MemoryRequest;
      Buffer->Next++;
      if (Buffer->Next>Buffer->Max) Buffer->Full=Yes;
      Buffer->Empty=No;
      if (CacheWaitingFor==Buffer->WaitingForFlag) CacheWaitingFor=Nothing;
      }

   }
```

```
/*****************************************************************************
**                                                              Page 10- 7  **
**                                 Buffer.c                                 **
**                                                                          **
**                                   View                                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       View returns a copy of the top record in the buffer without        **
**  altering the buffer.                                                    **
**                                                                          **
*****************************************************************************/


MemoryRequestType View(Buffer)

   BufferType *Buffer;

   {

   MemoryRequestType MemoryRequest;

   if (Buffer->Empty)
      {
      printf("\n\n   Tryed to View an empty buffer!\n\n");
      exit(1);
      }
   else
      {
      MemoryRequest=Buffer->MemoryRequest[0];
      }

   return(MemoryRequest);

   }
```

```
/*******************************************************************
**                                                      Page 10- 8  **
**                              Buffer.c                            **
**                                                                  **
**                               Clear                              **
**                                                                  **
**  Description:                                                    **
**                                                                  **
*       Clear removes all entrees in the buffer.                    **
**                                                                  **
*******************************************************************/


void Clear(Buffer)

   BufferType *Buffer;

   {
   Buffer->Next=0;
   Buffer->Full=No;
   Buffer->Empty=Yes;
   }
```

```
*****************************************************************************
*                                                                Page 10- 9  **
*                                 Buffer.c                                  **
*                                                                           **
*                                  Order                                    **
*                                                                           **
*  Description:                                                             **
*                                                                           **
*       Order sorts all of the entries in the buffer by priority such       **
*  that the highest priority (lowest priority number) is at the top.        **
*                                                                           **
*****************************************************************************/


oid Order(Buffer)

  BufferType *Buffer;

  {

  MemoryRequestType TmpMemoryRequest;
  YesNoType         Change=Yes;
  BufferSizeType    i;

  while (!(Buffer->Empty) && Change)
     {
     Change=No;
     for (i=Buffer->Next-1; i>0; i--)
        {
        if (Buffer->MemoryRequest[i].Priority<
            Buffer->MemoryRequest[i-1].Priority)
           {
           TmpMemoryRequest=Buffer->MemoryRequest[i];
           Buffer->MemoryRequest[i]=Buffer->MemoryRequest[i-1];
           Buffer->MemoryRequest[i-1]=TmpMemoryRequest;
           Change=Yes;
           }
        }
     }

  }
```

```
/*****************************************************************************
**                                                          Page 10-10  **
**                                Buffer.c                                **
**                                                                        **
**                                 Splice                                 **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       Splice is buffer utility that takes a one byte memory request and **
**  enters it into a buffer if the buffer does not already have the byte.  **
**  Splice will first searchthe ReadBuffer for the byte if it can't find a **
**  request in the buffer that contains the byte then it will search for a **
**  memory request that has data from the same block.  If one is found    **
**  then the request is modified to include the new read byte request.    **
**  If no suitable request can be found then Splice will add a one byte   **
**  memory request to the Buffer.                                         **
**                                                                        **
*****************************************************************************/


void Splice(Buffe ,Address,RequiredSize,Block,Priority)

   BufferType     *Buffer;
   AddressType    Address;
   SizeType       RequiredSize;
   SizeType       Block;
   PriorityType   Priority;

   {

   BufferSizeType    BufferIndex;
   YesNoType         FoundByte=No;
   AddressType       FrontAddress;
   AddressType       BackAddress;
   AddressType       CurrentBlockAddress=BlockAddress(Address);
   SizeType          NextSize;

   MemoryRequestType MemoryRequest;

   MemoryRequest.Address                = Address;
   MemoryRequest.Size                   = 1;
   MemoryRequest.RequiredSize           = 0;
   MemoryRequest.Block                  = Block;
   MemoryRequest.Priority               = Priority;
   MemoryRequest.AccessInProgress       = No;
   MemoryRequest.TimeToExecute          = 0;
   MemoryRequest.CompletionTimeEstimate = 0;
   if (RequiredSize>0) MemoryRequest.RequiredSize=1;
```

```
/*****************************************************************************
**                                                          Page 10-11   **
**                                Buffer.c                                **
**                                                                        **
**                                 Splice                                 **
**                                continued                               **
**                                                                        **
*****************************************************************************/


   if (!(Buffer->Empty))
      {
      for (BufferIndex=0; BufferIndex<Buffer->Next; BufferIndex++)
         {
         if (BlockAddress(Buffer->MemoryRequest[BufferIndex].Address)==
             CurrentBlockAddress)
            {
            NextSize=1;
            FrontAddress=Buffer->MemoryRequest[BufferIndex].Address;
            BackAddress =FrontAddress;
            while (FoundByte==No && NextSize<=BlockSize &&
                   NextSize<=(Buffer->MemoryRequest[BufferIndex].Size+1))
               {
               if (BackAddress==Address)
                  {
                  if (NextSize>Buffer->MemoryRequest[BufferIndex].Size)
                     Buffer->MemoryRequest[BufferIndex].Size=NextSize;
                  if (RequiredSize>0)
                     Buffer->MemoryRequest[BufferIndex].RequiredSize=NextSize;
                  if (Buffer->MemoryRequest[BufferIndex].Priority>Priority)
                     Buffer->MemoryRequest[BufferIndex].Priority=Priority;
                  FoundByte=Yes;
                  }
               if (FrontAddress==Address &&
                   Buffer->MemoryRequest[BufferIndex].AccessInProgress==No)
                  {
                  Buffer->MemoryRequest[BufferIndex].Size=NextSize;
                  if (RequiredSize>0)
                      Buffer->MemoryRequest[BufferIndex].RequiredSize++;
                  if (Buffer->MemoryRequest[BufferIndex].Priority>Priority)
                      Buffer->MemoryRequest[BufferIndex].Priority=Priority;
                  FoundByte=Yes;
                  }
               NextSize++;
               if (NextSize>Buffer->MemoryRequest[BufferIndex].Size)
                  FrontAddress--;
               BackAddress++;
               if (BlockAddress(FrontAddress)!=CurrentBlockAddress)
                  FrontAddress+=BlockSize;
               if (BlockAddress(BackAddress) !=CurrentBlockAddress)
                  BackAddress-=BlockSize;
               }
            if (Buffer->MemoryRequest[BufferIndex].Size==BlockSize &&
                Buffer->MemoryRequest[BufferIndex].AccessInProgress==No)
               {
               Buffer->MemoryRequest[BufferIndex].Address=RequestAddress;
               Buffer->MemoryRequest[BufferIndex].RequiredSize=RequestSize;
               }
            }
         }
      }
   if (FoundByte==No) Append(Buffer,&MemoryRequest);
   }
```

```
/*****************************************************************************
**                                                              Page 10-12  **
**                                 Buffer.c                                 **
**                                                                          **
**                                  Search                                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Search checks a buffer to see if it contains a byte addressed by   **
**  Address.                                                                **
**                                                                          **
*****************************************************************************/


YesNoType Search(Buffer,Address)

   BufferType      *Buffer;
   AddressType     Address;

   {

   BufferSizeType BufferIndex;
   AddressType    ByteAddress;
   AddressType    CurrentBlockAddress=BlockAddress(Address);
   SizeType       NoBytes;
   YesNoType      FoundByte=No;

   if (!(Buffer->Empty))
      {
      for (BufferIndex=0; BufferIndex<Buffer->Next; BufferIndex++)
         {
         if (BlockAddress(Buffer->MemoryRequest[BufferIndex].Address)==
             CurrentBlockAddress)
            {
            ByteAddress=Buffer->MemoryRequest[BufferIndex].Address;
            for (NoBytes=0;
                 NoBytes<Buffer->MemoryRequest[BufferIndex].Size;
                 NoBytes++)
               {
               if (ByteAddress==Address) FoundByte=Yes;
               ByteAddress++;
               if (BlockAddress(ByteAddress)!=CurrentBlockAddress)
                  ByteAddress-=BlockSize;
               }
            }
         }
      }

   return(FoundByte);

   }
```

```
/*****************************************************************************
**                                                              Page 10-13  **
**                                 Buffer.c                                 **
**                                                                          **
**                            UpdatingReadBuffer                            **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       UpdatingReadBuffer takes a byte of data provided by a CPU write    **
**  reqeust and checks to see if it is needed in the read buffer.  If the  **
**  byte is needed then the MemoryRequest is modified so that the byte     **
**  is no longer in the request.                                            **
**                                                                          **
*****************************************************************************/


YesNoType UpdatingReadBuffer(Address)

   AddressType Address;

   {

   AddressType    ByteAddress;
   AddressType    CurrentBlockAddress = BlockAddress(Address);
   BufferSizeType BufferIndex;
   YesNoType      FoundByte           = No;
```

```
/*****************************************************************************
**                                                                          **
**                                Buffer.c                                  **
**                                                                          **
**                            UpdatingReadBuffer                            **
**                                continued                                 **
**                                                                          **
*****************************************************************************/

   if (!(ReadBuffer.Empty))
      {
      for (BufferIndex=0; BufferIndex<ReadBuffer.Next; BufferIndex++)
         {
         if (BlockAddress(ReadBuffer.MemoryRequest[BufferIndex].Address)==
                CurrentBlockAddress
             && ReadBuffer.MemoryRequest[BufferIndex].AccessInProgress==No)
            {
            if (ReadBuffer.MemoryRequest[BufferIndex].Size>0)
               {
               ByteAddress=ReadBuffer.MemoryRequest[BufferIndex].Address +
                           ReadBuffer.MemoryRequest[BufferIndex].Size    - 1;
               if (ByteAddress==Address)
                  {
                  ReadBuffer.MemoryRequest[BufferIndex].Size--;
                  if (ReadBuffer.MemoryRequest[BufferIndex].RequiredSize>
                      ReadBuffer.MemoryRequest[BufferIndex].Size)
                      ReadBuffer.MemoryRequest[BufferIndex].RequiredSize=
                         ReadBuffer.MemoryRequest[BufferIndex].Size;
                  FoundByte=Yes;
                  }
               }

            if (ReadBuffer.MemoryRequest[BufferIndex].Size>0)
               {
               if (ReadBuffer.MemoryRequest[BufferIndex].Address==Address)
                  {
                  ReadBuffer.MemoryRequest[BufferIndex].Address++;
                  if (BlockAddress(ReadBuffer.MemoryRequest[BufferIndex].Address)
                      !=CurrentBlockAddress)
                     ReadBuffer.MemoryRequest[BufferIndex].Address-=BlockSize;
                  if (ReadBuffer.MemoryRequest[BufferIndex].Size   >0)
                      ReadBuffer.MemoryRequest[BufferIndex].Size--;
                  if (ReadBuffer.MemoryRequest[BufferIndex].RequiredSize>0)
                      ReadBuffer.MemoryRequest[BufferIndex].RequiredSize--;
                  FoundByte=Yes;
                  }
               }
            }
         }
      }

   RemoveZeroSizes(&ReadBuffer);

   return(FoundByte);

   }
```

```
/*****************************************************************************
**                                                              Page 10-15  **
**                                 Buffer.c                                 **
**                                                                          **
**                              RemoveZeroSizes                             **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       RemoveZeroSizes removes all entrees that have a zero size from     **
**  the buffer.                                                             **
**                                                                          **
*****************************************************************************/


void RemoveZeroSizes(Buffer)

   BufferType *Buffer;

   {

   BufferSizeType i=0;
   BufferSizeType j=0;

   while (j<Buffer->Next)
      {

      if (Buffer->MemoryRequest[j].Size==0 &&
          !Buffer->MemoryRequest[j].AccessInProgress)
         {
         j++;
         Buffer->Full=No;
         }
      else
         {
         Buffer->MemoryRequest[i]=Buffer->MemoryRequest[j];
         i++;
         j++;
         }

      }

   Buffer->Next=i;
   if (Buffer->Next==0) Buffer->Empty=Yes;

   }
```

```
/*****************************************************************************
**                                                                Page 10-16  **
**                                Buffer.c                                   **
**                                                                           **
**                             NoRequestsLeft                                **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       NoRequestsLeft returns Yes if there are no more requests left       **
**  in the buffer.                                                           **
**                                                                           **
*****************************************************************************/


YesNoType NoRequestsLeft(Buffer)

   BufferType *Buffer;

   {

   BufferSizeType i;

   for (i=0; i<Buffer->Next; i++)
      {
      if (Buffer->MemoryRequest[i].RequiredSize>0) return(No);
      }

   return(Yes);

   }
```

```
/*****************************************************************************
**                                                          Page 11- 0      **
**                                Array.c                                   **
**                                                                          **
**                            Part Of SACS 1.0                              **
**                      (StillAnother Cache Simulator)                      **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                              Page 11- 1  **
**                                 Array.c                                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Array.c contains all functions that relate to definition and       **
**  freeing, or allocation, and deallocation of arrays.                     **
**                                                                          **
**                              Table of Contents                           **
**                                                                          **
**          Cover Page ............................... Page 11- 1          **
**          List of Array.c Function Declarations ..... Page 11- 2          **
**          DefineArray1D() ........................... Page 11- 3          **
**          DefineArray2D() ........................... Page 11- 4          **
**          FreeArray1D() ............................. Page 11- 5          **
**          FreeArray2D() ............................. Page 11- 6          **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                              Page 11- 2  **
**                                Array.c                                   **
**                                                                          **
**                  List of Array.c Function Declarations                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "Array.c".                                                           **
**                                                                          **
*****************************************************************************/


int  *DefineArray1D();                                          /* Page 11- 3 */
int **DefineArray2D();                                          /* Page 11- 4 */
void  FreeArray1D();                                            /* Page 11- 5 */
void  FreeArray2D();                                            /* Page 11- 6 */
```

```
/*****************************************************************************
**                                                              Page 11- 3  **
**                                 Array.c                                  **
**                                                                          **
**                              DefineArray1D                               **
**  Description:                                                            **
**                                                                          **
**       DefineArray1D allocates memory large enough for a 1 dimensional    **
**  array of length Xmax, where each element has "size" bytes.              **
**                                                                          **
*****************************************************************************/


int *DefineArray1D(Xmax, size)

  unsigned Xmax;
  unsigned size;

  {

  int *Array;

  Array=(int*) calloc(Xmax,size);

  return(Array);

  }
```

```
/*****************************************************************************
**                                                          Page 11- 4  **
**                                Array.c                                **
**                                                                       **
**                             DefineArray2D                             **
**                                                                       **
**  Description:                                                         **
**                                                                       **
**       DefineArray2D allocates memory large enough for a 2 dimensional **
**  array Xmax, by Ymax, where each element has "size" bytes.            **
**                                                                       **
*****************************************************************************/


int **DefineArray2D(Xmax,Ymax,size)

  unsigned Xmax;
  unsigned Ymax;
  unsigned size;

  {

  int      **Array;
  unsigned x;

  Array=(int**) calloc(Xmax,size);

  for (x=0; x<Xmax; x++) Array[x]=(int*) calloc(Ymax,size);

  return(Array);

  }
```

```
/*****************************************************************************
**                                                          Page 11- 5   **
**                              Array.c                                   **
**                                                                        **
**                            FreeArray1D                                 **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       FreeArray1D deallocates the memory assigned to the 1 dimensional  **
**  array.                                                                **
**                                                                        **
*****************************************************************************/

void FreeArray1D(Array,Xmax)

  int *Array;

  {

  free(Array);

  }
```

```
/*****************************************************************************
**                                                              Page 11- 6  **
**                                 Array.c                                  **
**                                                                          **
**                               FreeArray2D                                **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       FreeArray2D deallocates the memory assigned to the 2 dimensional   **
**  array.                                                                  **
**                                                                          **
*****************************************************************************/


void FreeArray2D(Array,Xmax,Ymax)

  int      **Array;
  unsigned Xmax;
  unsigned Ymax;

  {

  unsigned x;

  for (x=0; x<Xmax; x++) free(Array[x]);

  free(Array);

  }
```

```
/*****************************************************************************
**                                                          Page 12- 0      **
**                              TestSACS.c                                  **
**                                                                          **
**                          Part Of SACS 1.0                                **
**                    (StillAnother Cache Simulator)                        **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, person-  or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                                Page 12- 1  **
**                                TestSACS.c                                **
**  Description:                                                            **
**                                                                          **
**       TestSACS randomly creates instructions and writes them to          **
**  "SACS.Dat".  The instrcutions are generated by first choosing           **
**  NoTestCases.  The Number of test cases to be used will always           **
**  be less than MaxNoTestCases. Then the TestCaseChoosen is picked         **
**  from TestCases.  TestCases represent different possible ways in         **
**  which an address trace could procede.  PredictedNoRead, HitsInTest,     **
**  and PredictedNoWriteHitsInTest, tells TestSACS how many hits it can     **
**  expect because, it used a specific test.                                **
**                                                                          **
**      Example:                                                            **
**                                                                          **
**         main()                                                           **
**            {                                                             **
**                                                                          **
**            unsigned long int NoLoadHits;                                 **
**            unsigned long int NoLoadRequests;                             **
**                                                                          **
**            TestSACS(NumberOfRequests, NumberOfHits);                     **
**                                                                          **
**            printf("\n\nNumber of read hits=%ul",NoReadHits);             **
**            printf("\n\nNumber of read requests=%ul",NoReadRequests);     **
**            printf("\n\nNumber of write hits=%ul",NoWriteHits);           **
**            printf("\n\nNumber of write requests=%ul",NoWriteRequests);   **
**                                                                          **
**            }                                                             **
**                                                                          **
**                              Table of Contents                           **
**                                                                          **
**          Cover Page ............................... Page 12- 1          **
**          List of definitions ...................... Page 12- 2          **
**          List of TestSACS.c Function Declarations .. Page 12- 3          **
**          TestCases ................................ Page 12- 4          **
**          PredictedNoReadHits ...................... Page 12- 5          **
**          PredictedNoWriteHits ..................... Page 12- 5          **
**          ChangeArguments() ........................ Page 12- 6          **
**          TestSACS() ............................... Page 12- 8          **
**             CreateInstructionSets() ............... Page 12- 9          **
**                Creating One Instruciton at a Time .. Page 12-10          **
**             ShufflingInstructionSets() ............ Page 12-12          **
**                CanBeSwitched() ...................... Page 12-14          **
**             WriteInstructionSet() ................. Page 12-15          **
**                                                                          **
*****************************************************************************/


#include<time.h>

#include "Global.h"
```

```
/*****************************************************************************
**                                                              Page 12- 2  **
**                                TestSACS.c                                **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       List of definitions.                                               **
**                                                                          **
*****************************************************************************/


#define MaxNoOfTestCases         3 /* Can be changed without other changes. */

#define NoTestCaseChoices       64 /*     // Need to change TestCases, and  */
#define NoLoadStoresInTestCases  7 /*     \\ PredictedNoHitsInTest          */

#define lrand() ((unsigned long) ((rand()*0x1000l+rand())*0x1000l+rand()))
```

```
/*****************************************************************************
**                                                                Page 12- 3  **
**                                TestSACS.c                                **
**                                                                          **
**                List of TestSACS.c Function Declarations                  **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of functions declarations within the file scope     **
**  of "TestSACS.c".                                                        **
**                                                                          **
*****************************************************************************/

void      ChangeArguments();                                  /* Page 12- 6 */
void      TestSACS();                                         /* Page 12- 8 */
void         CreateInstructionSets();                         /* Page 12- 9 */
void         ShufflingInstructionSets();                      /* Page 12-12 */
YesNoType       CanBeSwitched();                              /* Page 12-14 */
void         WriteInstructionSet();                           /* Page 12-15 */
```

```
/*****************************************************************************
**                                                                Page 12- 4  **
**                              TestSACS.c                                   **
**                                                                           **
**                              TestCases                                    **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       Loading Test Cases, the each number is an indexe for an array of    **
**  BlockAddressChoices.                                                     **
**                                                                           **
*****************************************************************************/


   int TestCases[NoTestCaseChoices][NoLoadStoresInTestCases]=
      {
      { 1, 1, 1, 1, 1, 1, 1}, { 1, 1, 1, 1, 1, 1, 2},
      { 1, 1, 1, 1, 1, 2, 1}, { 1, 1, 1, 1, 1, 2, 2},
      { 1, 1, 1, 1, 2, 1, 1}, { 1, 1, 1, 1, 2, 1, 2},
      { 1, 1, 1, 1, 2, 2, 1}, { 1, 1, 1, 1, 2, 2, 2},
      { 1, 1, 1, 2, 1, 1, 1}, { 1, 1, 1, 2, 1, 1, 2},
      { 1, 1, 1, 2, 1, 2, 1}, { 1, 1, 1, 2, 1, 2, 2},
      { 1, 1, 1, 2, 2, 1, 1}, { 1, 1, 1, 2, 2, 1, 2},
      { 1, 1, 1, 2, 2, 2, 1}, { 1, 1, 1, 2, 2, 2, 2},
      { 1, 1, 2, 1, 1, 1, 1}, { 1, 1, 2, 1, 1, 1, 2},
      { 1, 1, 2, 1, 1, 2, 1}, { 1, 1, 2, 1, 1, 2, 2},
      { 1, 1, 2, 1, 2, 1, 1}, { 1, 1, 2, 1, 2, 1, 2},
      { 1, 1, 2, 1, 2, 2, 1}, { 1, 1, 2, 1, 2, 2, 2},
      { 1, 1, 2, 2, 1, 1, 1}, { 1, 1, 2, 2, 1, 1, 2},
      { 1, 1, 2, 2, 1, 2, 1}, { 1, 1, 2, 2, 1, 2, 2},
      { 1, 1, 2, 2, 2, 1, 1}, { 1, 1, 2, 2, 2, 1, 2},
      { 1, 1, 2, 2, 2, 2, 1}, { 1, 1, 2, 2, 2, 2, 2},
      { 1, 2, 1, 1, 1, 1, 1}, { 1, 2, 1, 1, 1, 1, 2},
      { 1, 2, 1, 1, 1, 2, 1}, { 1, 2, 1, 1, 1, 2, 2},
      { 1, 2, 1, 1, 2, 1, 1}, { 1, 2, 1, 1, 2, 1, 2},
      { 1, 2, 1, 1, 2, 2, 1}, { 1, 2, 1, 1, 2, 2, 2},
      { 1, 2, 1, 2, 1, 1, 1}, { 1, 2, 1, 2, 1, 1, 2},
      { 1, 2, 1, 2, 1, 2, 1}, { 1, 2, 1, 2, 1, 2, 2},
      { 1, 2, 1, 2, 2, 1, 1}, { 1, 2, 1, 2, 2, 1, 2},
      { 1, 2, 1, 2, 2, 2, 1}, { 1, 2, 1, 2, 2, 2, 2},
      { 1, 2, 2, 1, 1, 1, 1}, { 1, 2, 2, 1, 1, 1, 2},
      { 1, 2, 2, 1, 1, 2, 1}, { 1, 2, 2, 1, 1, 2, 2},
      { 1, 2, 2, 1, 2, 1, 1}, { 1, 2, 2, 1, 2, 1, 2},
      { 1, 2, 2, 1, 2, 2, 1}, { 1, 2, 2, 1, 2, 2, 2},
      { 1, 2, 2, 2, 1, 1, 1}, { 1, 2, 2, 2, 1, 1, 2},
      { 1, 2, 2, 2, 1, 2, 1}, { 1, 2, 2, 2, 1, 2, 2},
      { 1, 2, 2, 2, 2, 1, 1}, { 1, 2, 2, 2, 2, 1, 2},
      { 1, 2, 2, 2, 2, 2, 1}, { 1, 2, 2, 2, 2, 2, 2}
      };
```

```
/*****************************************************************************
**                                                              Page 12- 5  **
**                                TestSACS.c                                **
**                                                                          **
**              PredictedNoReadHits, and PredictedNoWriteHits               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       No Hits Predicted for each test case.                              **
**                                                                          **
*****************************************************************************/


   int PredictedNoReadHitsInTest[NoTestCaseChoices]=
      {
      6, 5, 5, 4,
      5, 4, 4, 3,
      5, 4, 4, 3,
      4, 3, 3, 2,
      5, 4, 4, 3,
      4, 3, 3, 2,
      4, 3, 3, 2,
      3, 2, 2, 1,
      5, 4, 4, 3,
      4, 3, 3, 2,
      4, 3, 3, 2,
      3, 2, 2, 1,
      4, 3, 3, 2,
      3, 2, 2, 1,
      3, 2, 2, 1,
      2, 1, 1, 0
      };


   int PredictedNoWriteHitsInTest[NoTestCaseChoices]=
      {
      0, 0, 0, 1,
      0, 1, 1, 2,
      0, 1, 1, 2,
      1, 2, 2, 3,
      0, 1, 1, 2,
      1, 2, 2, 3,
      1, 2, 2, 3,
      2, 3, 3, 4,
      0, 1, 1, 2,
      1, 2, 2, 3,
      1, 2, 2, 3,
      2, 3, 3, 4,
      1, 2, 2, 3,
      2, 3, 3, 4,
      2, 3, 3, 4,
      3, 4, 4, 5
      };
```

```
/*****************************************************************************
**                                                                Page 12- 6  **
**                              TestSACS.c                                  **
**                                                                          **
**                            ChangeArguments                               **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       ChangeArguments, change the global variables in SACS that the      **
**  user can change.                                                        **
**                                                                          **
*****************************************************************************/

void ChangeArguments()

   {

   SizeType          WordSizeLimit          =  8;
   SizeType          WordsPerSubBlock       =  4;
   SizeType          NumberOfSubBlocksLimit =  4;
   SizeType          NumberOfBlocksLimit    = 32;
   AssociativityType AssociativityLimit     =  8;
   BufferSizeType    BufferSizeLimit        =  8;
   TimeType          TimeLimit              =  8;

   WordSize                      = (rand()%(WordSizeLimit-1))+1;
   SubBlockSize                  = WordSize*((rand()%WordsPerSubBlock)+1);
   BlockSize                     = SubBlockSize
                                   *((rand()%NumberOfSubBlocksLimit)+1);

   Associativity                 = (rand()%AssociativityLimit)+2;
   CacheSize                     = BlockSize*Associativity*
                                   ((rand()%NumberOfBlocksLimit)+1);

   ReadCacheAccessTime           = rand()%TimeLimit;
   ReadCacheHitTime              = rand()%TimeLimit;
   ReadCacheMissTime             = rand()%TimeLimit;
   WriteCacheAccessTime          = rand()%TimeLimit;
   WriteCacheHitTime             = rand()%TimeLimit;
   WriteCacheMissTime            = rand()%TimeLimit;
   MemoryAccessTime              = rand()%TimeLimit;
   MemoryTransferTime            = rand()%TimeLimit;
   BufferCacheAccessTime         = rand()%TimeLimit;

   ReadBufferSize                = (rand()%BufferSizeLimit)+1;
   WriteBufferSize               = (rand()%BufferSizeLimit)+1;

   BlockReplacementPolicy        = rand()%NumberOfReplacementPoliciesAvailable;
   WritePolicy                   = rand()%NumberOfWritePoliciesAvailable;
   WriteMissPolicy               = rand()%NumberOfWriteMissPoliciesAvailable;
   ReadForward                   = rand()%Unknown;
   CPUWaitsForCacheWrites        = rand()%Unknown;
   SearchBlockBuffer             = rand()%Unknown;
   UpdateReadBuffer              = rand()%Unknown;
   RemoveReadDuplicates          = rand()%Unknown;
   RemoveWriteDuplicates         = rand()%Unknown;

   ReadPriority                  = (rand()%(NoPriority-1))+1;
   WritePriority                 = (rand()%(NoPriority-1))+1;
   ReadForWriteAllocatePriority  = (rand()%(NoPriority-1))+1;
   WriteDirtyBlockPriority       = (rand()%(NoPriority-1))+1;
```

```
/*****************************************************************************
**                                                              Page 12- 7  **
**                              TestSACS.c                                  **
**                                                                          **
**                            ChangeArguments                               **
**                              continued                                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Ensuring that the new arguments are valid combinations.            **
**                                                                          **
*****************************************************************************/

   if (SearchBlockBuffer==No)  RemoveReadDuplicates=No;

   if (UpdateReadBuffer ==Yes) WordSize=SubBlockSize;

   }
```

```
/*****************************************************************************
**                                                              Page 12- 8  **
**                              TestSACS.c                                  **
**                                                                          **
**                               TestSACS                                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       TestSACS will create a test set, shuffle the instructions, and     **
**  write them out to "SACS.Dat".                                           **
**                                                                          **
*****************************************************************************/


void TestSACS(NumberOfRequests, NumberOfHits)

   ScoreType *NumberOfRequests;
   ScoreType *NumberOfHits;

   {

   char        Request             [MaxNoOfTestCases*NoTestCaseChoices+1];
   AddressType DataAddress         [MaxNoOfTestCases*NoTestCaseChoices+1];
   SizeType    Size                [MaxNoOfTestCases*NoTestCaseChoices+1];
   TimeType    TimeUntilNextRequest[MaxNoOfTestCases*NoTestCaseChoices+1];
   int         Imax=0;

   DisplayTestingHeader();

   CreateInstructionSets(Request, DataAddress, Size, TimeUntilNextRequest,
                         &Imax,
                         NumberOfRequests, NumberOfHits);

   ShufflingInstructionSets(Request, DataAddress, Size, TimeUntilNextRequest,
                            Imax);
   rewind(DataFile);

   WriteInstructionSet(Request, DataAddress, Size, TimeUntilNextRequest,
                       Imax);
   rewind(DataFile);
   EndOfDataFile=No;

   }
```

```
/*****************************************************************************
**                                                          Page 12- 9   **
**                              TestSACS.c                                **
**                                                                        **
**                         CreateInstructionSets                          **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       The instructions are created from a set of test cases.  The      **
**  the number of predicted hits for each case is stored in               **
**  PredictedNoHitsInTest.  CreateInstructionsSets randomly chooses the   **
**  NoOfTestCases to be used, and randomly selects the individual         **
**  TestCases.  The BlockAddressChoices for each test case is also chosen **
**  randomly.  The DataAddress, and Size of each instuction is chosen     **
**  randomly such that they are within the block chosen.  The NoLoadHits  **
**  is predicted by summing up all of the PredictedNoHitsInTest.          **
**                                                                        **
*****************************************************************************/


void CreateInstructionSets(Request,
                           DataAddress,
                           Size,
                           TimeUntilNextRequest,
                           Imax,
                           NumberOfRequests,
                           NumberOfHits)

   char        *Request;
   AddressType *DataAddress;
   SizeType    *Size;
   TimeType    *TimeUntilNextRequest;
   int         *Imax;
   ScoreType   *NumberOfRequests;
   ScoreType   *NumberOfHits;

   {

   TimeType    MaxTimeUntilNextRequest = 101;
   AddressType BlockAddressChoices[NumberOfRequestsAvailable];
   SizeType    SetChosen;

   int TestCaseIndex;
   int LoadIndex;
   int NoOfTestCases;
   int TestCaseChosen;
   int i,j,k;

   time_t t;
/* srand((unsigned) time(&t)); */   /* Uncomment to randomize each test run */
                                    /* otherwise every test run will be     */
                                    /* identical.  Leaving the seed         */
                                    /* commented out allows any errors      */
                                    /* found by -test to be revisited.      */

   NumberOfRequests[Read ]=0;
   NumberOfRequests[Write]=0;
   NumberOfHits    [Read ]=0;
   NumberOfHits    [Write]=0;
```

```
/*****************************************************************************
**                                                              Page 12-10  **
**                              TestSACS.c                                  **
**                                                                          **
**                        CreatedInstrutionSets                             **
**                              Continued                                   **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Creating one instruction at a time.                                **
**                                                                          **
*****************************************************************************/


   *Imax=0;
   NoOfTestCases=rand()%MaxNoOfTestCases+1;
   for (TestCaseIndex=0; TestCaseIndex<NoOfTestCases; TestCaseIndex++)
      {

      TestCaseChosen=rand()%NoTestCaseChoices;
      SetChosen=rand()%NumberOfSets;

      BlockAddressChoices[Read]=
                     ( ( (lrand()/(NoOfTestCases*NumberOfSets*BlockSize))
                        *NoOfTestCases+TestCaseIndex)
                     *NumberOfSets + SetChosen) * BlockSize;

      BlockAddressChoices[Write]=BlockAddressChoices[Read];
      if (rand()%2)
         BlockAddressChoices[Write]=
                     ( ( (lrand()/(NoOfTestCases*NumberOfSets*BlockSize))
                         *NoOfTestCases+TestCaseIndex)
                        *NumberOfSets + SetChosen) * BlockSize;

      if (BlockAddress(BlockAddressChoices[Read])!=BlockAddressChoices[Read])
         printf("Read is not a BlockAddress");
      if (BlockAddress(BlockAddressChoices[Write])!=BlockAddressChoices[Write])
         printf("Write is not a BlockAddress");

      if (Set(BlockAddressChoices[Read])!=SetChosen)
         printf("Read is not a good Set");

      if (Set(BlockAddressChoices[Write])!=SetChosen)
         printf("Write is not a good Set");

      if (BlockAddressChoices[Read]==BlockAddressChoices[Write])
         printf("%4dE",TestCaseChosen);
      else
         printf("%4dN",TestCaseChosen);
      if (TestCaseIndex%15==14) printf("\n");
```

```
/*****************************************************************************
**                                                          Page 12-11   **
**                              TestSACS.c                                **
**                                                                        **
**                         CreatedInstrutionSets                          **
**                               Continued                                **
**                                                                        **
*****************************************************************************/

      if (BlockAddressChoices[Read]==BlockAddressChoices[Write])
         {
         NumberOfHits[Read] +=PredictedNoReadHitsInTest[TestCaseChosen];
         NumberOfHits[Write]+=PredictedNoWriteHitsInTest[TestCaseChosen];

         LoadIndex=0;
         while (TestCases[TestCaseChosen][LoadIndex]==Write &&
                WriteMissPolicy!=WriteAllocate              &&
                LoadIndex<NoLoadStoresInTestCases)
            {
            NumberOfHits[Write]--;
            LoadIndex++;
            }
         if (TestCases[TestCaseChosen][0]==Read && TestCaseChosen!=0)
            NumberOfHits[Write]++;
         if (TestCases[TestCaseChosen][0]==Write &&
             WriteMissPolicy==WriteAllocate)
            NumberOfHits[Read]++;
         }
      else
         {
         NumberOfHits[Read] +=PredictedNoReadHitsInTest[TestCaseChosen];
         if (WriteMissPolicy==WriteAllocate)
            NumberOfHits[Write]+=PredictedNoWriteHitsInTest[TestCaseChosen];
         }

      for (LoadIndex=0; LoadIndex<NoLoadStoresInTestCases; LoadIndex++)
         {
         if (TestCases[TestCaseChosen][LoadIndex]==Read)
            {
            Request[*Imax]='r';
            NumberOfRequests[Read]++;
            }
         else
            {
            Request[*Imax]='w';
            NumberOfRequests[Write]++;
            }

         DataAddress[*Imax] =
            BlockAddressChoices[TestCases[TestCaseChosen][LoadIndex]];

         Size[*Imax]=lrand()%BlockSize+1;
         if (Size[*Imax]<BlockSize)
            DataAddress[*Imax] += lrand()%(BlockSize-Size[*Imax])+1;
         TimeUntilNextRequest[*Imax]=lrand()%MaxTimeUntilNextRequest;

         (*Imax)++;
          }
       }
    }
```

```
/*****************************************************************************
**                                                              Page 12-12   **
**                              TestSACS.c                                   **
**                                                                           **
**                        ShufflingInstructionSets                           **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       Shuffling Instruction Sets.                                         **
**                                                                           **
*****************************************************************************/


void ShufflingInstructionSets(Request,
                              DataAddress,
                              Size,
                              TimeUntilNextRequest,
                              Imax)

   char        *Request;
   AddressType *DataAddress;
   SizeType    *Size;
   TimeType    *TimeUntilNextRequest;
   int          Imax;

   {

   int          Jump;
   char         RequestTemp;
   AddressType  AddressTemp;
   SizeType     SizeTemp;
   TimeType     TimeTemp;
   int i,j,k;
```

```
/*****************************************************************************
**                                                                          **
**                              TestSACS.c                                  **
**                                                                          **
**                        ShufflingInstructionSets                          **
**                              continued                                   **
**                                                                          **
*****************************************************************************/


   for (i=1; i<Imax; i++)
      {

      Jump=Yes;

      for (j=i; j>0 && Jump==Yes; j--)
         {

         if (BlockAddress(DataAddress[j])==BlockAddress(DataAddress[j-1]))
            Jump=No;

         if ((rand()%(Associativity*NoLoadStoresInTestCases))==0)
            Jump=No;  /* Gives Uniform distrabution. */

         if (Jump==Yes)
            Jump=CanBeSwitched(DataAddress, j, Imax);

         if (Jump==Yes)
            {
            RequestTemp=Request[j-1];
            Request[j-1]=Request[j];
            Request[j]=RequestTemp;

            AddressTemp=DataAddress[j-1];
            DataAddress[j-1]=DataAddress[j];
            DataAddress[j]=AddressTemp;

            SizeTemp=Size[j-1];
            Size[j-1]=Size[j];
            Size[j]=SizeTemp;

            TimeTemp=TimeUntilNextRequest[j-1];
            TimeUntilNextRequest[j-1]=TimeUntilNextRequest[j];
            TimeUntilNextRequest[j]=TimeTemp;
            }

         }
      }
   }
```

```
/*****************************************************************************
**                                                          Page 12-14   **
**                              CanBeSwitched                             **
**                                                                        **
**  DESCRIPTION:                                                          **
**                                                                        **
**       Can InstructionAddress[i] be switched with Instruction[i-1].     **
**  If so return Yes, else return No.                                     **      **
**                                                                        **
*****************************************************************************/


YesNoType CanBeSwitched(DataAddress, Io, Imax)
   unsigned long int *DataAddress;
   int               Io;
   int               Imax;

   {

   int               i,j;
   YesNoType         Jump=Yes;
   YesNoType         NoJumped;
   YesNoType         JumpedBefore;
   unsigned long int AddressJumped[100];

   Jump=Yes;

   NoJumped=1;
   AddressJumped[0]=BlockAddress(DataAddress[Io]);
   i=Io-2;
   while (i>=0 && NoJumped<Associativity &&
          BlockAddress(DataAddress[i])!=BlockAddress(DataAddress[Io-1]))
      {
      JumpedBefore=No;
      for (j=0; j<NoJumped; j++)
         if (AddressJumped[j]==BlockAddress(DataAddress[i])) JumpedBefore=Yes;
      if (JumpedBefore==No && NoJumped<Associativity)
         AddressJumped[NoJumped++]=BlockAddress(DataAddress[i]);
      i--;
      }
   if (NoJumped>=Associativity) Jump=No;

   NoJumped=1;
   AddressJumped[0]=BlockAddress(DataAddress[Io-1]);
   i=Io+1;
   while (i<Imax && NoJumped<Associativity &&
          BlockAddress(DataAddress[i])!=BlockAddress(DataAddress[Io]))
      {
      JumpedBefore=No;
      for (j=0; j<NoJumped; j++)
         if (AddressJumped[j]==BlockAddress(DataAddress[i])) JumpedBefore=Yes;
      if (JumpedBefore==No && NoJumped<Associativity)
         AddressJumped[NoJumped++]=BlockAddress(DataAddress[i]);
      i++;
      }
   if (NoJumped>=Associativity) Jump=No;

   return(Jump);

   }
```

```
/*****************************************************************************
**                                                                Page 12-15  **
**                              TestSACS.c                                   **
**                                                                           **
**                         WriteInstructionSet                               **
**                                                                           **
**  Description:                                                             **
**                                                                           **
**       Write SACS.Dat one line at a time.                                  **
**                                                                           **
*****************************************************************************/


void WriteInstructionSet(Request,
                         DataAddress,
                         Size,
                         TimeUntilNextRequest,
                         Imax)

   char        *Request;
   AddressType *DataAddress;
   SizeType    *Size;
   TimeType    *TimeUntilNextRequest;
   int         Imax;

   {

   int i;

   for (i=0; i<Imax; i++)
      {
      fprintf(DataFile,"%c "   ,Request[i]);
      fprintf(DataFile,"%08lX ",DataAddress[i]);
      fprintf(DataFile,"%2u "  ,Size[i]);
      fprintf(DataFile,"%lu"   ,TimeUntilNextRequest[i]);
      fprintf(DataFile,"\n");
      }

   fprintf(DataFile,"End Of Trace\n\n");
   fprintf(DataFile,"If any instructions follow\n");
   fprintf(DataFile,"they were not used for the\n");
   fprintf(DataFile,"last run.                 \n");
   fprintf(DataFile,"\n");

   }
```

```
/*****************************************************************************
**                                                          Page 13- 0      **
**                              Checking.c                                  **
**                                                                          **
**                           Part Of SACS 1.0                               **
**                     (StillAnother Cache Simulator)                       **
**                                                                          **
**   Program Modified: 3/17/94                                              **
**   File Modified:    3/17/94                                              **
**                                                                          **
**   Author:  William G. Smith                                              **
**   Address:  Electrical Engineering Department                            **
**             Naval Postgraduate School                                    **
**             Monterey, CA 93940                                           **
**                                                                          **
**   Copyright 1994, William G. Smith                                       **
**                                                                          **
**   Permission to use, copy, modify, and distribute this software and      **
**   its documentation for any purpose and without fee is hereby granted    **
**   provided that the above copyright notice appears in all copies.  No    **
**   modified version of this program should be redistributed without the   **
**   authors consent.  William G. Smith makes no warranty or                **
**   representation, promise of guarantee, either expressed or implied,     **
**   with respect to this software's ability to produce valid results.      **
**   This program is provided "as is" any financial, personal or property   **
**   damage caused by the use of this program is the responsibility of the  **
**   user.                                                                  **
**                                                                          **
*****************************************************************************/
```

```
/*****************************************************************************
**                                                          Page 13- 1   **
**                              Checking.c                                **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       Checking.c contains all of the functions that relate to error    **
**  checking.  Note that an error could be raised anywhere.  The error    **
**  mesage will contain the procedure name in square brackets.  This      **
**  section contains the functions spicifically designed to check variable **
**  to see if they are consitant with each other, and if they are within  **
**  set boundarys.                                                        **
**                                                                        **
**                            Table of Contents                           **
**                                                                        **
**          Cover Page ............................... Page 13- 1        **
**          List of Cache.c Function Declarations ..... Page 13- 2        **
**          Checking() ............................... Page 13- 3        **
**             CheckingConstants() ................... Page 13- 4        **
**                PrintConstError() ................... Page 13-11        **
**             CheckingForValuesOutOfBounds() ......... Page 13-12        **
**                PrintTimeBoundaryError(); ........... Page 13-15        **
**                PrintScoreBoundaryError(); .......... Page 13-16        **
**                PrintSizeBoundaryError(); ........... Page 13-17        **
**                PrintEnumBoundaryError(); ........... Page 13-18        **
**             CheckingForInconsistencies() ........... Page 13-19        **
**                PrintTotalTimeError() ............... Page 13-21        **
**                PrintTotalScoreError() .............. Page 13-22        **
**          CheckingPredictions() ................... Page 13-23        **
**             PrintScorePredictionError() ............ Page 13-24        **
**          PrintTimePredictionError() ............... Page 13-25        **
**                                                                        **
*****************************************************************************/

#include "Global.h"
```

```
/*****************************************************************************
**                                                              Page 13- 2  **
**                                Checking.c                                **
**                                                                          **
**                  List of Checking.c Function Declarations                **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       This is a list of function declarations within the file scope      **
**  of "Checking.c".                                                        **
**                                                                          **
*****************************************************************************/


void Checking();                                          /* Page 13- 3 */
void    CheckingConstants();                              /* Page 13- 4 */
void       PrintConstError();                             /* Page 13-11 */
void    CheckingForValuesOutOfBounds();                   /* Page 13-12 */
void       PrintTimeBoundaryError();                      /* Page 13-15 */
void       PrintScoreBoundaryError();                     /* Page 13-16 */
void       PrintSizeBoundaryError();                      /* Page 13-17 */
void       PrintEnumBoundaryError();                      /* Page 13-18 */
void    CheckingForInconsistencies();                     /* Page 13-19 */
void       PrintTotalTimeError();                         /* Page 13-21 */
void       PrintTotalScoreError();                        /* Page 13-22 */
void CheckingPredictions();                               /* Page 13-23 */
void    PrintScorePredictionError();                      /* Page 13-24 */
void PrintTimePredictionError();                          /* Page 13-25 */
```

```
/*****************************************************************************
**                                                                Page 13- 3  **
**                                Checking.c                                **
**                                                                          **
**                                 Checking                                 **
**                                                                          **
**  Description:                                                            **
**                                                                          **
**       Checking checks the global variables to insure that constants      **
**  remain constant, and Values are in bounds, als that there are no       **
**  inconsistencies.                                                        **
**                                                                          **
*****************************************************************************/

void Checking()

   {

   CheckingConstants(No);
   CheckingForValuesOutOfBounds();
   CheckingForInconsistencies();

   }
```

```
/*****************************************************************************
**                                                              Page 13- 4  **
**                                Checking.c                               **
**                                                                         **
**                             CheckingConstants                           **
**                                                                         **
**  Description:                                                           **
**                                                                         **
**       Checking global constants to insure that they do not change,      **
**  unless they are being Reset.                                           **
**                                                                         **
*****************************************************************************/


void CheckingConstants(Reset)

   YesNoType Reset;

   {

   static CacheSizeType              CacheSizeCopy;
   static SizeType                   BlockSizeCopy;
   static SizeType                   SubBlockSizeCopy;
   static AssociativityType          AssociativityCopy;
   static SizeType                   WordSizeCopy;

   static TimeType                   ReadCacheAccessTimeCopy;
   static TimeType                   ReadCacheHitTimeCopy;
   static TimeType                   ReadCacheMissTimeCopy;
   static TimeType                   WriteCacheAccessTimeCopy;
   static TimeType                   WriteCacheHitTimeCopy;
   static TimeType                   WriteCacheMissTimeCopy;

   static TimeType                   MemoryAccessTimeCopy;
   static TimeType                   MemoryTransferTimeCopy;
   static TimeType                   BufferCacheAccessTimeCopy;

   static BufferSizeType             ReadBufferSizeCopy;
   static BufferSizeType             WriteBufferSizeCopy;

   static BlockReplacementPolicyType BlockReplacementPolicyCopy;
   static WritePolicyType            WritePolicyCopy;
   static WriteMissPolicyType        WriteMissPolicyCopy;
   static YesNoType                  ReadForwardCopy;
   static YesNoType                  CPUWaitsForCacheWritesCopy;
   static YesNoType                  SearchBlockBufferCopy;
   static YesNoType                  UpdateReadBufferCopy;
   static YesNoType                  RemoveReadDuplicatesCopy;
   static YesNoType                  RemoveWriteDuplicatesCopy;

   static PriorityType               ReadPriorityCopy;
   static PriorityType               WritePriorityCopy;
   static PriorityType               ReadForWriteAllocatePriorityCopy;
   static PriorityType               WriteDirtyBlockPriorityCopy;
   static PriorityType               NoPriorityCopy;

   static YesNoType                  CheckCopy;

   static YesNoType                  KeyBoardIOCopy;
   static char                       *DataFileNameCopy;

   static HistogramIndexType         ScreenHistogramMaxIndexCopy;
   static HistogramIndexType         FileHistogramMaxIndexCopy;
```

```
/*****************************************************************************
**                                                                          **
**                                Checking.c                                **
**                                                                          **
**                             CheckingConstants                            **
**                                 continued                                **
**                                                                          **
*****************************************************************************/

   static SizeType            NumberOfBlocksCopy;
   static SizeType            NumberOfSubBlocksCopy;
   static SizeType            NumberOfSetsCopy;

   static AddressType         *CacheBlockAddressCopy;
   static TimeType            *LastCacheBlockAccessTimeCopy;

   static SizeType            *CacheNextBlockCopy;

   static YesNoType           **CacheValidBitCopy;
   static YesNoType           **CacheDirtyBitCopy;

   static TimeType            **RequestTimeHistogramCopy;
   static TimeType            **StallTimeHistogramCopy;

   static TimeType            *TotalRequestTimeCopy;
   static TimeType            *TotalStallTimeCopy;

   static ScoreType           *NumberOfAccessesCopy;
   static ScoreType           *NumberOfCacheHitsCopy;
   static ScoreType           *NumberOfBufferHitsCopy;
   static ScoreType           *PredictedNumberOfAccessesCopy;
   static ScoreType           *PredictedNumberOfHitsCopy;

   static FILE                *DataFileCopy;
```

```
   if (Reset)

      {

      CacheSizeCopy                    = CacheSize;
      BlockSizeCopy                    = BlockSize;
      SubBlockSizeCopy                 = SubBlockSize;
      AssociativityCopy                = Associativity;
      WordSizeCopy                     = WordSize;

      ReadCacheAccessTimeCopy          = ReadCacheAccessTime;
      ReadCacheHitTimeCopy             = ReadCacheHitTime;
      ReadCacheMissTimeCopy            = ReadCacheMissTime;
      WriteCacheAccessTimeCopy         = WriteCacheAccessTime;
      WriteCacheHitTimeCopy            = WriteCacheHitTime;
      WriteCacheMissTimeCopy           = WriteCacheMissTime;

      MemoryAccessTimeCopy             = MemoryAccessTime;
      MemoryTransferTimeCopy           = MemoryTransferTime;
      BufferCacheAccessTimeCopy        = BufferCacheAccessTime;

      ReadBufferSizeCopy               = ReadBufferSize;
      WriteBufferSizeCopy              = WriteBufferSize;

      BlockReplacementPolicyCopy       = BlockReplacementPolicy;
      WritePolicyCopy                  = WritePolicy;
      WriteMissPolicyCopy              = WriteMissPolicy;
      ReadForwardCopy                  = ReadForward;
      CPUWaitsForCacheWritesCopy       = CPUWaitsForCacheWrites;
      SearchBlockBufferCopy            = SearchBlockBuffer;
      UpdateReadBufferCopy             = UpdateReadBuffer;
      RemoveReadDuplicatesCopy         = RemoveReadDuplicates;
      RemoveWriteDuplicatesCopy        = RemoveWriteDuplicates;

      ReadPriorityCopy                 = ReadPriority;
      WritePriorityCopy                = WritePriority;
      ReadForWriteAllocatePriorityCopy = ReadForWriteAllocatePriority;
      WriteDirtyBlockPriorityCopy      = WriteDirtyBlockPriority;
      NoPriorityCopy                   = NoPriority;

      CheckCopy                        = Check;

      KeyBoardIOCopy                   = KeyBoardIO;
      DataFileNameCopy                 = DataFileName;

      ScreenHistogramMaxIndexCopy      = ScreenHistogramMaxIndex;
      FileHistogramMaxIndexCopy        = FileHistogramMaxIndex;
```

```
    NumberOfBlocksCopy              = NumberOfBlocks;
    NumberOfSubBlocksCopy           = NumberOfSubBlocks;
    NumberOfSetsCopy                = NumberOfSets;

    CacheBlockAddressCopy           = CacheBlockAddress;
    LastCacheBlockAccessTimeCopy    = LastCacheBlockAccessTime;

    CacheNextBlockCopy              = CacheNextBlock;

    CacheValidBitCopy               = CacheValidBit;
    CacheDirtyBitCopy               = CacheDirtyBit;

    RequestTimeHistogramCopy        = RequestTimeHistogram;
    StallTimeHistogramCopy          = StallTimeHistogram;

    TotalRequestTimeCopy            = TotalRequestTime;
    TotalStallTimeCopy              = TotalStallTime;

    NumberOfAccessesCopy            = NumberOfAccesses;
    NumberOfCacheHitsCopy           = NumberOfCacheHits;
    NumberOfBufferHitsCopy              = NumberOfBufferHits;
    PredictedNumberOfAccessesCopy   = PredictedNumberOfAccesses;
    PredictedNumberOfHitsCopy       = PredictedNumberOfHits;

    DataFileCopy                    = DataFile;

    }
```

```
   if (!(Reset))

      {

      if( CacheSizeCopy                    != CacheSize)
          PrintConstError("CacheSize");
      if( BlockSizeCopy                    != BlockSize)
          PrintConstError("BlockSize");
      if( SubBlockSizeCopy                 != SubBlockSize)
          PrintConstError("SubBlockSize");
      if( AssociativityCopy                != Associativity)
          PrintConstError("Associativity");
      if( WordSizeCopy                     != WordSize)
          PrintConstError("WordSize");
      if( ReadCacheAccessTimeCopy          != ReadCacheAccessTime)
          PrintConstError("ReadCacheAccessTime");
      if( ReadCacheHitTimeCopy             != ReadCacheHitTime)
          PrintConstError("ReadCacheHitTime");
      if( ReadCacheMissTimeCopy     != ReadCacheMissTime)
          PrintConstError("ReadCacheMissTime");
      if( WriteCacheAccessTimeCopy         != WriteCacheAccessTime)
          PrintConstError("WriteCacheAccessTime");
      if( WriteCacheHitTimeCopy            != WriteCacheHitTime)
          PrintConstError("WriteCacheHitTime");
      if( WriteCacheMissTimeCopy           != WriteCacheMissTime)
          PrintConstError("WriteCacheMissTime");
      if( MemoryAccessTimeCopy             != MemoryAccessTime)
          PrintConstError("MemoryAccessTime");
      if( MemoryTransferTimeCopy           != MemoryTransferTime)
          PrintConstError("MemoryTransferTime");
      if( BufferCacheAccessTimeCopy        != BufferCacheAccessTime)
          PrintConstError("BufferCacheAccessTime");
      if( ReadBufferSizeCopy               != ReadBufferSize)
          PrintConstError("ReadBufferSize");
      if( WriteBufferSizeCopy              != WriteBufferSize)
          PrintConstError("WriteBufferSize");
      if( BlockReplacementPolicyCopy       != BlockReplacementPolicy)
          PrintConstError("BlockReplacementPolicy");
      if( WritePolicyCopy                  != WritePolicy)
          PrintConstError("WritePolicy");
      if( WriteMissPolicyCopy              != WriteMissPolicy)
          PrintConstError("WriteMissPolicy");
      if( ReadForwardCopy                  != ReadForward)
          PrintConstError("ReadForward");
      if( CPUWaitsForCacheWritesCopy       != CPUWaitsForCacheWrites)
          PrintConstError("CPUWaitsForCacheWrites");
      if( SearchBlockBufferCopy            != SearchBlockBuffer)
          PrintConstError("SearchBlockBuffer");
      if( UpdateReadBufferCopy             != UpdateReadBuffer)
          PrintConstError("UpdateReadBuffer");
```

Checking.c

CheckingConstants
continued

```
      if( RemoveReadDuplicatesCopy          != RemoveReadDuplicates)
          PrintConstError("RemoveReadDuplicates");
      if( RemoveWriteDuplicatesCopy         != RemoveWriteDuplicates)
          PrintConstError("RemoveWriteDuplicates");
      if( ReadPriorityCopy                  != ReadPriority)
          PrintConstError("ReadPriority");
      if( WritePriorityCopy                 != WritePriority)
          PrintConstError("WritePriority");
      if( ReadForWriteAllocatePriorityCopy != ReadForWriteAllocatePriority)
          PrintConstError("ReadForWriteAllocatePriority");
      if( WriteDirtyBlockPriorityCopy       != WriteDirtyBlockPriority)
          PrintConstError("WriteDirtyBlockPriority");
      if( NoPriorityCopy                    != NoPriority)
          PrintConstError("NoPriority");
      if( CheckCopy                         != Check)
          PrintConstError("Check");
      if( KeyBoardIOCopy                    != KeyBoardIO)
          PrintConstError("KeyBoardIO");
      if( DataFileNameCopy                  != DataFileName)
          PrintConstError("DataFileName");
      if( ScreenHistogramMaxIndexCopy       != ScreenHistogramMaxIndex)
          PrintConstError("ScreenHistogramMaxIndex");
      if( FileHistogramMaxIndexCopy         != FileHistogramMaxIndex)
          PrintConstError("FileHistogramMaxIndex");
```

```
/*****************************************************************************
**                                                                          **
**                                Checking.c                                **
**                                                                          **
**                             CheckingConstants                            **
**                                 continued                                **
**                                                                          **
*****************************************************************************/

      if( NumberOfBlocksCopy                 != NumberOfBlocks)
          PrintConstError("NumberOfBlocks");
      if( NumberOfSubBlocksCopy              != NumberOfSubBlocks)
          PrintConstError("NumberOfSubBlocks");
      if( NumberOfSetsCopy                   != NumberOfSets)
          PrintConstError("NumberOfSets");
      if( CacheBlockAddressCopy              != CacheBlockAddress)
          PrintConstError("CacheBlockAddress");
      if( LastCacheBlockAccessTimeCopy       != LastCacheBlockAccessTime)
          PrintConstError("LastCacheBlockAccessTime");
      if( CacheNextBlockCopy                 != CacheNextBlock)
          PrintConstError("CacheNextBlock");
      if( CacheValidBitCopy                  != CacheValidBit)
          PrintConstError("CacheValidBit");
      if( CacheDirtyBitCopy                  != CacheDirtyBit)
          PrintConstError("CacheDirtyBit");
      if( RequestTimeHistogramCopy           != RequestTimeHistogram)
          PrintConstError("RequestTimeHistogram");
      if( StallTimeHistogramCopy             != StallTimeHistogram)
          PrintConstError("StallTimeHistogram");
      if( TotalRequestTimeCopy               != TotalRequestTime)
          PrintConstError("TotalRequestTime");
      if( TotalStallTimeCopy                 != TotalStallTime)
          PrintConstError("TotalStallTime");
      if( NumberOfAccessesCopy               != NumberOfAccesses)
          PrintConstError("NumberOfAccesses");
      if( NumberOfCacheHitsCopy              != NumberOfCacheHits)
          PrintConstError("NumberOfCacheHits");
      if( NumberOfBufferHitsCopy             != NumberOfBufferHits)
          PrintConstError("NumberOfBufferHitsCopy");
      if( PredictedNumberOfAccessesCopy      != PredictedNumberOfAccesses)
          PrintConstError("PredictedNumberOfAccesses");
      if( PredictedNumberOfHitsCopy          != PredictedNumberOfHits)
          PrintConstError("PredictedNumberOfHits");
      if( DataFileCopy                       != DataFile)
          PrintConstError("Datafile");

      }

   }
```

```
/*****************************************************************************
/*                                                                Page 13-11  **
/*                                Checking.c                                **
/*                                                                          **
/*                              PrintConstError                             **
/*                                                                          **
/*****************************************************************************/

void PrintConstError(VariableName)

   char *VariableName;

   {

   printf("\n\nError in [CheckingConstants]");
   printf("  \n%s did not remain constant.\n", VariableName);
   exit(0);

   }
```

```
/*****************************************************************************
**                                                          Page 13-12   **
**                                Checking.c                              **
**                                                                        **
**                        CheckingForValuesOutOfBounds                    **
**                                                                        **
**  Description:                                                          **
**                                                                        **
**       CheckingForValuesOutOfBounds checks all bounded global variables **
**  to see if they fall within their prescribed boundaries.               **
**                                                                        **
*****************************************************************************/


void CheckingForValuesOutOfBounds()

   {
   TimeType  MaxTime =10000000001;
   ScoreType MaxScore=10000000001;

   SizeType            BlockIndex;
   SizeType            SubBlockIndex;
   SizeType            SetIndex;
   RequestType         RequestIndex;
   CacheWaitingForType StallIndex;

   if (Time<0
      || Time>MaxTime)
      PrintTimeBoundaryError("Time",Time,01,MaxTime);

   if (  CacheWaitingFor<Nothing
      || CacheWaitingFor>NumberOfCacheWaitingForsAvailable)
      PrintEnumBoundaryError("CacheWaitingFor",CacheWaitingFor,
                             Nothing, NumberOfCacheWaitingForsAvailable);

   if (  MemoryWaitingFor<NothingTwo
      || MemoryWaitingFor>NumberOfMemoryWaitingForsAvailable)
      PrintEnumBoundaryError("MemoryWaitingFor",MemoryWaitingFor,
                             NothingTwo, NumberOfMemoryWaitingForsAvailable);

   if (  BlockWaitingFor<NothingThree
      || BlockWaitingFor>NumberOfBlockWaitingForsAvailable)
      PrintEnumBoundaryError("BlockWaitingFor",BlockWaitingFor,
                             NothingThree, NumberOfBlockWaitingForsAvailable );

   if (  CacheHit<No
      || CacheHit>Unknown)
      PrintEnumBoundaryError("CacheHit",CacheHit,No,Unknown);

   if (  CacheBusy<No
      || CacheBusy>Yes)
      PrintEnumBoundaryError("CacheBusy",CacheBusy,No,Yes);

   if (  Request<None
      || Request>NumberOfRequestsAvailable)
      PrintEnumBoundaryError("Request",Request,
                             None,NumberOfRequestsAvailable);

   if (  LastRequest<None
      || LastRequest>NumberOfRequestsAvailable)
      PrintEnumBoundaryError("LastRequest",Request,
                             None,NumberOfRequestsAvailable);
```

```
/*****************************************************************************
**                                                                          **
**                                Checking.c                                **
**                                                                          **
**                        CheckingForValuesOutOfBounds                      **
**                                continued                                 **
**                                                                          **
*****************************************************************************/


   if (  RequestSize<0
      || RequestSize>BlockSize)
      PrintSizeBoundaryError("RequestSize", RequestSize, 0, BlockSize);

   if (  RequestBlockNumber<0
      || RequestBlockNumber>=NumberOfBlocks)
      {
      PrintSizeBoundaryError("RequestBlockNumber",RequestBlockNumber,
                             0,NumberOfBlocks);
      }

   if (  TimeOfNextRequest<0
      || TimeOfNextRequest>MaxTime)
      PrintTimeBoundaryError("TimeOfNextRequest",TimeOfNextRequest,
                            0l,MaxTime);

   for (SetIndex=0; SetIndex<NumberOfSets; SetIndex++)
      if (  CacheNextBlock[SetIndex]<0
         || CacheNextBlock[SetIndex]>=NumberOfBlocks)
         PrintSizeBoundaryError("CacheNextBlock",CacheNextBlock[SetIndex],
                                0,NumberOfBlocks);

   for (BlockIndex=0; BlockIndex<NumberOfBlocks; BlockIndex++)
      {
      for (SubBlockIndex=0; SubBlockIndex<NumberOfSubBlocks; SubBlockIndex++)
         {
         if (  CacheValidBit[BlockIndex][SubBlockIndex]<0
            || CacheValidBit[BlockIndex][SubBlockIndex]>1)
            PrintEnumBoundaryError("CacheValidBit",
                                   CacheValidBit[BlockIndex][SubBlockIndex],
                                   0,1);

         if (  CacheDirtyBit[BlockIndex][SubBlockIndex]<0
            || CacheDirtyBit[BlockIndex][SubBlockIndex]>1)
            PrintEnumBoundaryError("CacheDirtyBit",
                                   CacheDirtyBit[BlockIndex][SubBlockIndex],
                                   0,1);
         }
      }

   for (RequestIndex=0; RequestIndex<NumberOfRequestsAvailable; RequestIndex++)
      if (  TotalRequestTime[RequestIndex]<0
         || TotalRequestTime[RequestIndex]>Time)
         PrintTimeBoundaryError("TotalRequestTime",
                                TotalRequestTime[RequestIndex],
                                0l,Time);
```

```
/*****************************************************************************
**                                                                          **
**                                Checking.c                                **
**                                                                          **
**                        CheckingForValuesOutOfBounds                      **
**                                continued                                 **
**                                                                          **
*****************************************************************************/


   for (StallIndex=0;
        StallIndex<NumberOfCacheWaitingForsAvailable;
        StallIndex++)
      if (  TotalStallTime[StallIndex]<0
         || TotalStallTime[StallIndex]>Time)
         PrintTimeBoundaryError("TotalStallTime",
                                TotalStallTime[StallIndex],
                                01,Time);

   if (  TotalNumberOfAccesses<0
      || TotalNumberOfAccesses>MaxScore)
      PrintScoreBoundaryError("TotalNumberOfAccesses",TotalNumberOfAccesses,
                              01, MaxScore);

   if (  TOA<0
      || TOA>MaxTime)
      PrintTimeBoundaryError("TOA", TOA, 01, MaxTime);

   if (  TOD<0
      || TOD>MaxTime)
      PrintTimeBoundaryError("TOD", TOD, 01, MaxTime);

   }
```

```
/*****************************************************************************
**                                                                Page 13-15  **
**                                Checking.c                                **
**                                                                          **
**                          PrintTimeBoundaryError                          **
**                                                                          **
*****************************************************************************/

void PrintTimeBoundaryError(VariableName, Value, LowLimit, HighLimit)

   char *VariableName;
   TimeType Value;
   TimeType LowLimit;
   TimeType HighLimit;

   {

   printf("\n\nError found in [CheckingForValuestOutOfBounds]");
   printf("  \n%s is out of prescribed bounds.",VariableName);

   printf("\n\n     The value was ............. ");
   PrintTime(Value);

   printf("  \n     The low limit was ......... ");
   PrintTime(LowLimit);

   printf("  \n     The high limit was ........ ");
   PrintTime(HighLimit);

   printf("\n\n");

   DiscrepancyFound=Yes;

   }
```

```
/*****************************************************************************
**                                                              Page 13-16  **
**                                Checking.c                               **
**                                                                         **
**                           PrintScoreBoundaryError                       **
**                                                                         **
*****************************************************************************/

void PrintScoreBoundaryError(VariableName, Value, LowLimit, HighLimit)

   char      *VariableName;
   ScoreType Value;
   ScoreType LowLimit;
   ScoreType HighLimit;

   {

   printf("\n\nError found in [CheckingForValuestOutOfBounds]");
   printf("  \n%s is out of prescribed bounds.",VariableName);

   printf("\n\n     The value was ............. ");
   PrintScoreCentered(Value);

   printf("  \n     The low limit was ......... ");
   PrintScoreCentered(LowLimit);

   printf("  \n     The high limit was ........ ");
   PrintScoreCentered(HighLimit);

   printf("\n\n");

   DiscrepancyFound=Yes;

   }
```

```
/*****************************************************************************
**                                                                Page 13-17  **
**                                Checking.c                                **
**                                                                          **
**                          PrintSizeBoundaryError                          **
**                                                                          **
*****************************************************************************/

void PrintSizeBoundaryError(VariableName, Value, LowLimit, HighLimit)

   char *VariableName;
   SizeType Value;
   SizeType LowLimit;
   SizeType HighLimit;

   {

   printf("\n\nError found in [CheckingForValuestOutOfBounds]");
   printf("  \n%s is out of prescribed bounds.",VariableName);

   printf("\n\n     The value was ............. ");
   PrintSize(Value);

   printf("  \n     The low limit was ......... ");
   PrintSize(LowLimit);

   printf("  \n     The high limit was ........ ");
   PrintSize(HighLimit);

   printf("\n\n");

   DiscrepancyFound=Yes;

   }
```

```
/*****************************************************************************
**                                                              Page 13-18  **
**                                Checking.c                                **
**                                                                          **
**                          PrintEnumBoundaryError                          **
**                                                                          **
*****************************************************************************/

void PrintEnumBoundaryError(VariableName, Value, LowLimit, HighLimit)

   char *VariableName;
   int  Value;
   int  LowLimit;
   int  HighLimit;

   {

   printf("\n\nError found in [CheckingForValuestOutOfBounds]");
   printf("  \n%s is out of prescribed bounds.",VariableName);

   printf("\n\n     The value was ............. ");
   printf("%d",Value);

   printf("  \n     The low limit was ......... ");
   printf("%d",LowLimit);

   printf("  \n     The high limit was ........ ");
   printf("%d",HighLimit);

   printf("\n\n");

   DiscrepancyFound=Yes;

   }
```

```
/*****************************************************************************
**                                                              Page 13-19  **
**                                Checking.c                                **
**                                                                          **
**                        CheckingForInconsistencies()                      **
**                                                                          **
*****************************************************************************/


void CheckingForInconsistencies()

   {

   CacheWaitingForType CacheWaitingForIndex;
   RequestType         RequestIndex;
   TimeType            TotalTime;

   ScoreType          SumOfAccesses;
   HistogramIndexType HistogramIndex;

   ScoreType          PredictedNumberOfWordsReadFromMemory;

   TotalTime=0;
   for (CacheWaitingForIndex=Nothing;
        CacheWaitingForIndex<NumberOfCacheWaitingForsAvailable;
        CacheWaitingForIndex++)
      TotalTime+=TotalStallTime[CacheWaitingForIndex];

   if (TotalTime!=Time)
      PrintTotalTimeError(Time,TotalTime,"Stalls");
```

```
/*****************************************************************************
**                                                                          **
**                                Checking.c                                **
**                                                                          **
**                        CheckingForInconsistencies()                      **
**                                                                          **
*****************************************************************************/


   TotalTime=0;
   for (RequestIndex=Nothing;
        RequestIndex<NumberOfRequestsAvailable;
        RequestIndex++)
      TotalTime+=TotalRequestTime[RequestIndex];


   if (TotalTime!=Time)
      PrintTotalTimeError(Time,TotalTime,"Requests");

   for (RequestIndex=Read;
        RequestIndex<NumberOfRequestsAvailable;
        RequestIndex++)
      {

      SumOfAccesses = 0;

      for (HistogramIndex=0;
           HistogramIndex<FileHistogramMaxIndex;
           HistogramIndex++)
         SumOfAccesses+=RequestTimeHistogram[RequestIndex][HistogramIndex];

      if (SumOfAccesses!=NumberOfAccesses[RequestIndex])
         PrintTotalScoreError(NumberOfAccesses[RequestIndex],
                              SumOfAccesses,
                              RequestString[RequestIndex]);

      }

   if (CacheWaitingFor==Nothing  && ReadBuffer.Empty==Yes &&
       UpdateReadBuffer==No      && SearchBlockBuffer==No &&
       RemoveReadDuplicates==Yes && WriteMissPolicy==WriteAround )
      {
      PredictedNumberOfWordsReadFromMemory=
         (NumberOfAccesses[Read]
          -NumberOfCacheHits[Read]
          -NumberOfBufferHits[Read])*BlockSize/WordSize;
      if (PredictedNumberOfWordsReadFromMemory!=
          TotalNumberOfWordsReadFromMemory)
         PrintTotalScoreError(PredictedNumberOfWordsReadFromMemory,
                              TotalNumberOfWordsReadFromMemory,
                              "Read Misses");
      }

   }
```

```
/*****************************************************************************
**                                                                Page 13-21  **
**                                Checking.c                                **
**                                                                          **
**                            PrintTotalTimeError                           **
**                                                                          **
*****************************************************************************/


void PrintTotalTimeError(TimeValue, TotalTimeValue, VariableName)

   TimeType TimeValue;
   TimeType TotalTimeValue;
   char     *VariableName;

   {

   printf("\n\nError found in [CheckingForInconsistencies] the total sum");
   printf("  \nof %s times does not equal the actual time.", VariableName);

   printf("\n\n     Total time was equal to ... ");
   PrintTime(TimeValue);

   printf("  \n     The sumation of %s", VariableName);
   printf("  \n     times was ................. ");
   PrintTime(TotalTimeValue);

   printf("\n\n");

   DiscrepancyFound=Yes;

   }
```

```
/*****************************************************************************
**                                                              Page 13-22  **
**                                Checking.c                                **
**                                                                          **
**                           PrintTotalScoreError                           **
**                                                                          **
*****************************************************************************/


void PrintTotalScoreError(TotalScoreValue, SumScoreValue, VariableName)

   ScoreType TotalScoreValue;
   ScoreType SumScoreValue;
   char      *VariableName;

   {

   printf("\n\nError found in [CheckingForInconsistencies] the total for");
   printf("  \n%s does not equal the summation.", VariableName);

   printf("\n\n     Total number of %s accesses", VariableName);
   printf("  \n     was equal to ............. ");
   PrintScoreCentered(TotalScoreValue);

   printf("  \n     The sumation of %s request histogram", VariableName);
   printf("  \n     was equal to ............. ");
   PrintScoreCentered(SumScoreValue);

   printf("\n\n");

   DiscrepancyFound=Yes;

   }
```

```
/*****************************************************************************
**                                                              Page 13-23  **
**                                Checking.c                                **
**                                                                          **
**                            CheckingPredictions                           **
**                                                                          **
*****************************************************************************/


void CheckingPredictions()

   {


   if (PredictedNumberOfAccesses[Read]!=NumberOfAccesses[Read])

      PrintScorePredictionError(PredictedNumberOfAccesses[Read],
                                NumberOfAccesses[Read],
                                "Read Accesses");

   if (PredictedNumberOfAccesses[Write]!=NumberOfAccesses[Write])

      PrintScorePredictionError(PredictedNumberOfAccesses[Write],
                               NumberOfAccesses[Write],
                               "Write Accesses");
   if (DiscrepancyFound==No)
      {
      if (PredictedNumberOfHits[Read] != NumberOfCacheHits[Read]
                                       + NumberOfBufferHits[Read] &&
          BlockReplacementPolicy==LRU                             &&
          SearchBlockBuffer==Yes && RemoveReadDuplicates==Yes)

         PrintScorePredictionError(PredictedNumberOfHits[Read],
                                   NumberOfCacheHits[Read]
                                   +NumberOfBufferHits[Read],
                                   "Read Hits");

      if (PredictedNumberOfHits[Write] != NumberOfCacheHits[Write]
                                        + NumberOfBufferHits[Write] &&
          BlockReplacementPolicy==LRU && WriteMissPolicy==WriteAllocate &&
          WritePolicy==WriteThrough)

         PrintScorePredictionError(PredictedNumberOfHits[Write],
                                   NumberOfCacheHits[Write]
                                   +NumberOfBufferHits[Write],
                                   "Write Hits");

      }

   }
```

```
/*****************************************************************************
**                                                                Page 13-24  **
**                                Checking.c                                **
**                                                                          **
**                           PrintPredictionError                           **
**                                                                          **
*****************************************************************************/


void PrintScorePredictionError(PredictedValue, ActualValue, VariableName)

   ScoreType PredictedValue;
   ScoreType ActualValue;
   char *VariableName;

   {

   printf("\n\nError found in [CheckingPredictions] when trying to predict %s",
         VariableName);
   printf("\n\n     The predicted value was ... ");
   PrintScoreCentered(PredictedValue);

   printf("  \n     The actual value was ...... ");
   PrintScoreCentered(ActualValue);

   printf("\n\n");

   DiscrepancyFound=Yes;

   }
```

```
/*****************************************************************************
**                                                                Page 13-25  **
**                                Checking.c                                **
**                                                                          **
**                          PrintTimePredictionError                        **
**                                                                          **
*****************************************************************************/


void PrintTimePredictionError(PredictedValue,
                              ActualValue,
                              RequestName,
                              ProcedureName)

   TimeType PredictedValue;
   TimeType ActualValue;
   char     *RequestName;
   char     *ProcedureName;

   {

   printf("\n\nError found in [%s] when trying to predict time to complete %s",
          ProcedureName, RequestName);

   printf("\n\n     The predicted value was ... ");
   PrintScoreCentered(PredictedValue);

   printf("  \n     The actual value was ...... ");
   PrintScoreCentered(ActualValue);

   printf("\n\n");

   DiscrepancyFound=Yes;

   }
```

BIBLIOGRAPHY

1. Jouppi, N.P., "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *The 17th Annual Symposium on Computer Architecture*, IEEE Computer Society Press, p. 364-373, May, 1990.

2. Hill, M. D., *Aspects of Cache Memory and Instruction Buffer Performance*. Ph.D. Thesis, University of California, Berkeley, 1987.

3. Smith, A. J., "Bibliography and Readings on CPU Cache Memories," *Computer Architecture News* v. 14-1, p. 22-42, January 1986.

4. Smith, A. J., "Second Bibliography on Cache Memories," v. 19-4, p. 154-182, June 1991.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library, Code 52 Naval Postgraduate School Monterey, CA 93943	2
3.	Chairman, Code EC Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943	1
4.	Douglas J. Fouts, Code EC/Fs Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943	2
5.	Shridhar B. Shukla, Code EC/Sh Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943	1
6.	Amr M. Zaky, Code CS/Za Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
7.	Diane Beckman PO Box 252 James Town, NY 14702	3